AD NUMBER

AD469503

LIMITATION CHANGES

TO:

Approved for public release; distribution is unlimited.

FROM:

**Distribution authorized to U.S. Gov't. agencies and their contractors;
Administrative/Operational Use; MAY 1963. Other requests shall be referred to Office of Naval Research, Washington, DC 20360.**

AUTHORITY

onr memo 7 Jan 1966

SECURITY MARKING

The classified or limited status of this report applies to each page, unless otherwise marked.
Separate page printouts MUST be marked accordingly.

THIS DOCUMENT CONTAINS INFORMATION AFFECTING THE NATIONAL DEFENSE OF THE UNITED STATES WITHIN THE MEANING OF THE ESPIONAGE LAWS, TITLE 18, U.S.C., SECTIONS 793 AND 794. THE TRANSMISSION OR THE REVELATION OF ITS CONTENTS IN ANY MANNER TO AN UNAUTHORIZED PERSON IS PROHIBITED BY LAW.

NOTICE: When government or other drawings, specifications or other data are used for any purpose other than in connection with a definitely related government procurement operation, the U. S. Government thereby incurs no responsibility, nor any obligation whatsoever; and the fact that the Government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data is not to be regarded by implication or otherwise as in any manner licensing the holder or any other person or corporation, or conveying any rights or permission to manufacture, use or sell any patented invention that may in any way be related thereto.

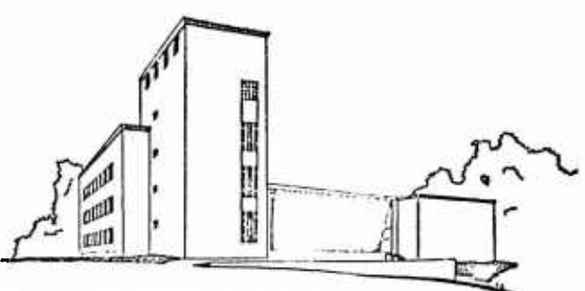
Defense Documentation Center

Defense Supply Agency

Cameron Station • Alexandria, Virginia

Carnegie Institute of Technology

Pittsburgh 13, Pennsylvania



GRADUATE SCHOOL of INDUSTRIAL ADMINISTRATION

William Larimer Mellon, Founder

O.N.R. Research Memorandum No. 113

(14) ONR-RM-113

THE SCHEDULING OF LARGE PROJECTS WITH LIMITED RESOURCES

(10) by

Jerome D. Wiest

(15) Nonr 760(24),
(16) NR047-011

May 15, 1962

Graduate School of Industrial Administration

Carnegie Institute of Technology

Pittsburgh 13, Pennsylvania

This paper was written as part of the contract, "Planning and Control of Industrial Operations," with the Office of Naval Research, at the Graduate School of Industrial Administration, Carnegie Institute of Technology. Reproduction of this paper in whole or in part is permitted for any purpose of the United States Government. Contract ONR-23

THE SCHEDULING OF LARGE PROJECTS WITH LIMITED RESOURCES

A Thesis

Submitted in partial fulfillment of the requirements

for the degree of Doctor of Philosophy

at the Carnegie Institute of Technology

by

Jerome Douglas Wiest

May, 1963

ACKNOWLEDGMENTS

The work reported here was accomplished with the generous support of the Ford Foundation, which provided Predoctoral Fellowships in 1960-62 and a Doctoral Dissertation Fellowship in 1962-63, and the Office of Naval Research, whose grants to the Graduate School of Industrial Administration supported (among several projects) my research during two summers and assisted in other ways. I am very grateful to both organizations.

The faculty committee under whom the thesis was written was a continual source of helpful suggestions. Professor G. L. Thompson, Chairman, fostered my interest in the topic; his steady guidance and stimulation were invaluable. Professors G. L. Bach, M. B. Nicholson, and F. M. Tonge provided a diversity of talents and viewpoints that I much appreciated. Their thoughtful comments are reflected in many ways throughout the thesis.

Several others could be mentioned who influenced my work at various stages. In particular, I would like to thank Ferdinand K. Levy, a fellow student. My research had its origin in some ideas we jointly developed during the summer of 1961; and in many discussions of my work we have had since then, he has been an interested and stimulating critic.

On my wife, Yvonne, fell the concurrent responsibilities of typing the thesis through its several drafts and final copy, managing a household and a husband, and caring for our new daughter, Merrilee, born in March of this year. (Moreover,

she successfully completed the numerous, partially-ordered jobs of these large projects without the aid of a computer, which speaks well of her own heuristic scheduling methods!) Throughout the years we have spent at Carnegie Tech, she has been a constant source of encouragement and inspiration, for which I am most grateful. That my work moved steadily to its completion is due in no small part to her sympathetic support.

C O N T E N T S

Chapter 4

Chapter 5

Chapter 6

LIST OF FIGURES

LIST OF TABLES

Chapter 1

LARGE PROJECT SCHEDULING: THE PROBLEM

Introduction

Large projects, as a class of human endeavor, have tested men's organizing abilities at least since the time of the Tower of Babel and Noah's Ark. Modern logistics problems have an ancient predecessor in the formidable project Moses undertook of planning the delivery of the Israelites from Egypt. One is impressed, in reading the history of civilization, that some of the most important periods of history are associated with the completion of large projects--from the building of the Egyptian pyramids to the explosion of the first atom bomb. The rapid advances of science and technology in our own age have led to a great acceleration in large project activities--evidenced, for example, by the construction of numerous dams, bridges, highway systems, and (more spectacularly) by the development and launching of space satelites.

Given such a long history of human involvement in large projects, it seems somewhat remarkable that, until quite recently, comparatively little has been written on the subject of large project scheduling. Perhaps the growing size and complexity of space-age projects has intensified the need for better methods of planning and scheduling such activities, which might explain the greatly increased attention devoted to this subject during the past three or four years. And computer technology has made easier the handling of large amounts of data associated with large projects. But one still wonders why researchers--at least in this century--have not earlier found the problems of large project scheduling an interesting area for study. Whatever the reason, the importance of the problem is presently evidenced by the widespread interest in it; and the modest amount of progress made thus far leaves it

still a fruitful area for research.

In the present volume, we have chosen to study, within the complex (and largely unstructured) field of large project management, the problem of project scheduling. We will not be concerned, for example, with questions of project design or technology, nor will we discuss implementation of a scheduling system or (in an explicit manner) the problems of project control. Further, we will deal with the case of certainty--that is, we will work with single job times rather than probability distributions or PERT-type estimates.

The specific goals of this volume are two-fold:

1) To develop a conceptual framework for the problem of large project scheduling, and to extend the concepts of critical path analysis to the general case of limited resources; and

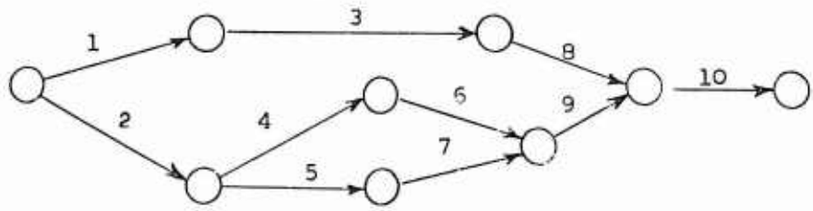
2) To develop and test some computer models for scheduling large projects with limited resources, drawing on the concepts developed in 1).

Definition of Large Project

The nature of large projects is evident, in part, from the examples we have cited, to which we might add the construction of buildings and plants, large maintenance projects (e.g., a turn-around in an oil refinery, in which the refinery is shut down for a few day for numerous repairs and alterations), research and engineering design projects, production of large, special-order equipment (e.g., power generating equipment), and so forth. Usually such projects are one-of-a kind, which means that schedules must be tailor-made for each project. Large projects typically consist of several hundred (or thousand) separate but technologically related jobs or activities. That is, the jobs are

partially ordered by predecessor-successor relationships; some jobs must be performed in a given sequence while others may be performed in parallel. Consider, for example, the following "project graph" of a simple project:

Figure 1



Each arrow represents a job or activity that requires certain resources and a given time to be completed. The connections of arrows and nodes indicate predecessor-successor relations. For example, job 1 is an immediate predecessor of job 3 and must be completed before 3 can begin. Jobs 4 and 5 are immediate successors of 2 and immediate predecessors of 6 and 7, respectively, and so forth. The completion of 10 marks the end of the project. A single due-date is of interest--the finish date of the entire project.

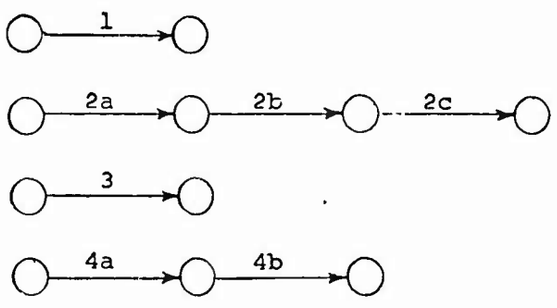
Structurally, the size of a project is not its dominant characteristic, of course. By defining a project simply as a collection of independent, partially-ordered jobs, we extend the application of our present analysis to projects of any size. Indeed, most of our illustrative examples throughout (as in Figure 1 above) involve trivially small projects. But it is the size of large projects that makes them interesting and worthy of our analysis. When we speak of the "large project problem," we mean to emphasize not only the structural aspects of a project but also the complexities that result from its size.

We should also note that our categorization of "project" and "job" is relative to one's point of view. Consider the Navy's task of scheduling ship repairs. From a top-level viewpoint, each ship to be repaired comprises a job, and the project consists of scheduling all the ship (jobs) that need repair in a given period of time. At the shipyard level, each ship which arrives is a project, and each major item to be repaired on the ship is a job. At an even lower level, such a job (e.g., overhauling the ship's engine) becomes a project with many separate, smaller jobs that must be scheduled. At each of these "levels of indenture," the essential structure of a project exists as we have defined it and our analysis is equally applicable. Thus the "size" of a project refers essentially to the number of jobs it contains, rather than to some physical or monetary measure of the project's importance.

The Large Project Problem vs. the Job-Shop Problem

The large project problem differs in several respects from the job-shop scheduling problem, in which the jobs, each of which may comprise one or more activities, are technologically independent (or are assumed to be), and the due-date of each job is of concern. A simple "job-shop graph" (comparable to the above project graph) might appear as follows:

Figure 2



Of the four jobs, 1 and 3 have a single activity, while 2 and 4 have several activities which must be performed in the sequence shown. A typical job-shop problem, of course, would involve hundreds or thousands of such jobs and activities. We will later have more to say about the relationship of the large-project and job-shop problems; suffice it now to observe that the former differs from the latter by virtue of the partial ordering of project jobs and the single due-date characteristic.

The Large Project Problem vs. the Line Balancing Problem

From a managerial viewpoint, the large project problem differs considerably from the line balancing problem. The latter is concerned with repetitive operations and large numbers of identical products, with the possibility of production for inventory, as one would find in mass-production type industries. On the other hand, a large project is, by definition, a one-of-a kind, single "product" effort. No attempt is made to group jobs or tasks into work stations, since the sequence of such jobs would be performed only once. Such efforts at sequencing in a mass production industry would likely be worth while because of large production runs. Thus line balancing is concerned with rate of production (units per time period). As an additional difference, each job in a project may require a different resource (skilled operator, machine, etc.), while the line balancing problem as ordinarily formulated implicitly assumes all operators are capable of performing any task at any work station.

Combinatorial Problems

In certain respects, however, the line balancing problem resembles both the large project problem and the job-shop problem. A similar directed graph representation may be made of the former, in which the arrows represent elemental tasks making up the assembly operations, and the node connections display precedence relationships among the elemental tasks (see Tonge [45]).[1] Thus Figure 1 might illustrate an assemply problem in which arrow 2 represents an elemental task requiring a given operations time per unit of product, which must be completed before elemental tasks 4 and 5 are performed on the same unit, and so forth. The problem is to assign tasks to work stations in such a way as to minimize the number of such work stations given the constraints of task times, ordering, and production rate. As Tonge notes, the line balancing and job-shop problems (and we might add, the large project problem) are representative of a class of combinatorial problems in which elements of a set are to be ordered or grouped according to some criterion. In job shop scheduling, groups of jobs are assigned to machines in such a way to observe sequencing and to minimize time past due-dates. In the large project problem, jobs are ordered in time in such a way that resources required do not exceed those available in any time period; the object is to minimize the project length given limited resources and the ordering constraints among the jobs.

---

1 The directed graph representation employed by Tonge is actually the reverse of that described above--i.e., nodes are elemental tasks and arrows represent precedence relationships. A project graph could also be drawn in this manner, of course (see Levy, Thompson, and Wiest [26]). While such a method of drawing a project graph avoids the necessity for "dummy jobs" (as described, for example, by Kelley [21]), the method used throughout this volume was chosen because it has certain advantages for purposes of illustration and because it has become so firmly established and widely accepted since the advent of PERT (Project Evaluation and Review Technique) and CPM (Critical Path Method).

One may conceive of the line balancing problem as one of ordering the elemental tasks along a time scale marked off into blocks determined by the desired production rate (e.g., a block would be two minutes if the production rate is 30 items/hour). Tasks may not be split between time blocks, and ordering constraints must be observed. The object is to minimize the number of time blocks (work stations), which amounts to assigning tasks in such a way as to maximize the proportion of time used for tasks in all of the blocks.[1]

An attempt to structure the large project problem in a similar manner points out the differences in the two problems. Since the project is performed just once, the concept of time blocks established by a production rate has no meaning. But we might conceive of resource blocks consisting of the amount of a resource available during one day (or whatever the smallest scheduling period may be). If jobs were all just one day long, they could be ordered along the resource scale in such a way that precedence relations were observed and resource requirements of jobs grouped into a resource block would never exceed the amount available in the block. The object would be to minimize the number of such blocks.

The difficulty with this formulation, of course, is that jobs are usually more than one day long. Additional complications arise from the fact that projects usually involve many resource types, jobs may require several resources, resources may vary over the schedule period (the resource "blocks" would be of different sizes), and resource requirements on many jobs

---

1 The formulation is due to Bowman [4].

are variable. Thus, as examples of combinatorial problems, line balancing and large project scheduling resemble each other only superficially.

Structure of the Large Project Problem

We turn now to examining the structure of large projects and the important constraints associated with them. The project scheduler, in assigning a start time to each job in a project, must consider

1) The partial ordering of jobs (i.e., no job may be started until all of its predecessors have been completed);

2) The resource requirements of each job (e.g., crew size, machine type, etc.);

3) The time required to complete the job (which often is a function of the resource requirement--i.e., the more resources assigned to a job, the less time it takes);

4) Resource limitations during any time period (which may be varied by hiring, lay-offs, overtime, etc.);

5) A due-Date or projected completion time for the entire project (sometimes with penalties attached for failure to meet it);

6) Other projects which may overlap the time period of the project being scheduled and thus compete for resources.

We could make our list much longer and more detailed, of course, but these are among the major constraints facing the project scheduler. Their relative importance may vary according to the nature of the project and the organizations involved. For example, general contractors scheduling a construction project typically are less concerned about resource limitations than with meeting the contract due-date. Construction workers--carpenters,

brick layers, etc.--are often obtained in any desired number through union hiring halls, and the contractor does not have to worry much about costs of hiring, firing and training. Resource limits, in other words, can be varied quite inexpensively. At the opposite extreme, Naval shipyards find it expensive, because of Civil Service regulations and labor contract requirements, to vary their shop crews to any great extent. Once a man is hired, it is difficult to release him. Thus shop crews tend to be relatively stable in size--usually large enough to handle peak manpower loads. The Navy, therefore, is concerned with smoothing out manpower requirements--lowering the peaks and filling the valleys--so as to obtain more efficient utilization of shop crews. The same is usually true of manufacturing enterprises engaged in the production of large items (e.g., hydro-electric generators). Costs of hiring, training, and layoffs (including "bumping," unemployment insurance, supplementary unemployment benefits) make many firms reluctant to vary work force levels to any great extent; although the possibility of overtime and (in some cases) subcontracting give a measure of flexibility to resource limits. Other examples may be cited where the limits on resources are even more inflexible--as, for example, where resources are already used extensively and overtime or subcontracting cannot be employed; where certain skilled workers are scarce and firms can hire no more (nor do they desire to lose the ones they have); where projects are intermittent and, by themselves, do not justify varying the existing work force (as in the month-end project of closing the accounting records in large firms).

Criterion Function

Because of the above differences in project types, the same criteria for a "good" schedule would not necessarily apply in every situation. Consider these three cases:

1) A contractor building a missile "site" for launching ICBM's works under a strict deadline and is quite willing to hire enough men to assure meeting the due date. Thus his goal is a schedule which optimizes resource levels--e.g., minimizes manpower costs (wages, idle time, overtime,, etc.), given a fixed due date.

2) At the other extreme is a project manager who cannot vary his resource levels; he wishes to find the schedule that minimizes the project length given fixed resources.

3) Other projects may fall in between these extremes; the manager desires to find some combination of resource levels and due date that will minimize resource costs, overhead costs, and penalties for exceeding the due date. The latter problem is the most difficult, of course, because it has the fewest constraints. The multi-dimensional space of possible schedules which must be searched is much larger due to the greater number of variables.

It is possible, of course, to write a criterion function that would apply to all three cases. If it includes all relevant costs, then the first case could be represented by assigning an extremely high penalty to extending the project past the due date. Variations in resource levels would thus be less expensive to explore than changes in due date. Likewise, in the second case, high costs attached to increases in resources would lead to the consideration, instead, of changes in due date. And in the third case, cost parameters would be such as to permit variations in

both due date and resource levels. Thus by altering the cost parameters in our criterion function, we can make it applicable to various project types or situations and reduce a possible multiplicity of functionals to the simple criterion, "minimize costs." This enables us to give a concise and general statement to the scheduling problem with which we are concerned.

The Problem: A Definition

Given a) a project consisting of a known, partially-ordered set of jobs, and b) limited resources with which to complete the jobs, find the schedule of job start times and crew assignments that minimizes all costs associated with the project.

We should immediately state that our goals are more modest than the above paragraph implies. We do not seek for a scheduling procedure that guarantees an optimum solution; we will be content with good solutions. In a later chapter we deal with the question, 'What is a good schedule?" Suffice it now to state that we seek an improvement over present scheduling procedures. The problem with which we are dealing is immense. Even modest sized projects have an enormous number of possible solutions, and there are no analytical techniques which can feasibly be applied. At times we may find it advisable, if not necessary, to simplify the problem by imposing constraints that narrow the range of possible solutions. Yet we intend to preserve as many of the essential characteristics of the project scheduling problem as possible, so that we end up with a procedure that can deal with a "real world" problem rather than an overly simplified and abstracted version of it. With the use of mathematical tools and the computational power of a modern digital computer, our goal is to develop scheduling procedures or models that are:

1) Sufficiently rich to take account of important job data (manpower requirements, time spans--with possible adjustments by varying the manpower, shop or skill requirements, and technological ordering) and shop characteristics (resource limits, regular and overtime labor costs, overhead costs);

2) Sufficiently general to be applicable to projects of different types or characteristics (and their associated criterion functions), and to multi-project situations in which each project has a unique due date (thus including, conceptually, the job-shop scheduling problem);

3) Computationally feasible for projects of reasonable size;

4) Superior, in an economic sense, to present methods of scheduling.

Chapter 2

CURRENT PRACTICES AND PROPOSALS IN LARGE PROJECT SCHEDULING

In this chapter we will explore and comment on the literature--traditional and current--dealing with topics related to large project scheduling. We will then examine an analytic formulation of the large project problem and finally present our argument for a heuristic approach to its solution.

Traditional Literature

Many large projects are carried on by manufacturing enterprises (e.g., makers of large turbo-electric generators, steel work prefabrications for bridges and buildings, large units of mining equipment, ships, etc.). Yet one looks with little success through books on manufacturing and production management for some explicit exposition of large project planning and scheduling. They generally deal instead with a related problem: intermittent manufacturing, the essential characteristic of which is "the quantity of any product made on any one order" (Moore [34], p. 21).

The dividing line between "large project" and "intermittent manufacturing," however, is a hazy one (Moore describes "special projects of gigantic size" as "intermittent manufacturing carried to its extreme"); hence the literature dealing with intermittent production scheduling is of interest to us. The traditional approach found in almost every book on production scheduling is to decentralize the assigning of specific start times of jobs in a project (or order). Moore, after describing the advantages of centralized scheduling (namely, better coordination and control), notes that the numerous, detailed directives necessary in intermittent manufacturing lead to decentralization of some production control work ([34], p. 54; see also [1, 24, 46]). The production

and sales departments determine what is to be produced at some aggregate level, and detailed planning and scheduling is often left to the foremen and in some cases, to the workers. Daily or weekly progress reports are made out by the foremen for the central office, which often finds it necessary to employ "stock chasers" or "expediters" to push delinquent jobs which threaten to delay completion of the project. "Order scheduling is concerned more with the setting of deadline dates than with setting exact time assignments for operations," the latter being established by foremen or local dispatchers essentially on the basis of urgency of need for items and availability of resources. Graphic tools, such as Gantt charts, are often used to keep track of resource assignments over a period of time and for measuring actual against planned progress (see Alford and Bangs [1, p.110]).

Thus only a preliminary or rough schedule of the project (order) is made centrally, while the more refined and detailed schedules are worked out by those nearer the operating level. The more decentralized the scheduling, the more cushion or slack time must be allowed at the operating level for juggling start dates within precedence requirements. This method of absorbing uncertainty has its costs in uneven shop loading, higher in-process inventories, and more distant completion dates. Decentralized scheduling also makes more difficult the prediction of manpower requirements. The decentralized scheduler has little means of determining the effects of his decisions on manpower loads of subsequent shops. Larger than necessary work forces and excessive idle time are often the result.

These disadvantages of decentralized scheduling have long been recognized, of course; but the problems associated with

centralized scheduling of large projects have, until recently, seemed intractable. The data-handling problem alone is immense. Projects typically contain 200 to 2,000 or more jobs. (Many job shops engaged in intermittent manufacturing have more than 5,000 jobs in process at one time.) Resource groups (shops) usually number between 8 and 20 or more, and the scheduling horizon extends in most cases beyond 150 days or time periods. A central scheduler faces a tremendously complex task if he attempts to establish a start date for each job in such a way as to observe job requirements (shop, number of men, machines, time, etc.), job sequence relationships, resource limitations (which may be changed by hiring, overtime, etc.), due date, and some criterion function such as minimal manpower requirements.

Manpower loads on various shops (or skill groups) typically are quite uneven. Production for inventory is, of course, unavailable as a method of smoothing peaks and valleys. Hence the scheduler must use the more difficult device of juggling jobs backwards or forwards, within the constraints of technological ordering, resource availability, and due date. The number of possible schedules he can devise is astronomical[1] and of course beyond his ability to explore. Hence the dilemma: over-all smoothing of manpower (and other resource requirements) can best be done, conceptually,

---

1 For example, assume a modest project of 200 jobs, consisting of 10 independent chains of 20 jobs linearly ordered, with one critical chain 100 days long and the other chains having 10 per cent slack. We can calculate the number of possible schedules by first counting for each chain the number of ways 10 "units" of slack can be distributed among 20 "units" of jobs. The answer is

$$\binom{30}{10,\ 20} = \frac{30}{10!\ 20!} = 1.48 \times 10^8$$

Thus, the total number of schedules for the whole project is $(1.48 \times 10^8)^9 = 3.4 \times 10^{73}$.

by a central scheduler; but the mass of data and the complex relationships of jobs in large projects are usually beyond his human capabilities to handle. Most often the result is a sub-optimum compromise: rough scheduling, with lots of cushion between jobs, is done centrally, while detailed schedules are generated at the operating level.[1]

Recent Developments in Large Project Scheduling

More recent efforts to solve various scheduling problems have drawn on new (or newly applied) mathematical techniques, the computational power of large digital computers, and various heuristic devices. Analytical solutions to the line balancing problem, for example, have been published by Jackson [19], who devised an enumerative algorithm, and Bowman [4], who developed

---

1 Admittedly, there are other valid reasons for decentralized scheduling apart from the problems of data processing. Job times are often rough estimates; actual work time may be more or less than expected. Resources available may also change unexpectedly, when men become sick, machines break down, material shortages develop, and so on. Decentralized scheduling, it is argued, is more flexible; a foreman can respond to unusual circumstances faster than can a production control man at head office (even if he has a computer at his finger tips) by juggling the start times and crews assigned to jobs on hand. But the ability to juggle jobs at the operating level is not inconsistent, necessarily, with centralized scheduling. In fact, if a central scheduler can adequately process the job data (e.g., by means of a computer model), he can do better than provide the foreman with large amounts of cushion in the schedule: he can tell the foreman the likely effects of delaying each of the jobs--which jobs have the most "slack" and can be delayed the longest without interrupting due dates, and which are critical and must be expedited. Much of the cushion built into schedules at present is not just to allow the foreman more flexibility in meeting unforseen contingencies, but rather it reflects the uncertainties of a scheduling system that cannot accurately predict daily (or hourly) resource needs for the number of jobs that have to be scheduled, and match them against resource availabilities.

a linear programming model. The computational requirements of both would strain the capacity of the largest computers on even moderate-sized problems. A heuristic approach--more flexible and practical than the present analytical methods--is described by Tonge [44].

In the job-shop scheduling problem (more closely related to large project scheduling than is the line balancing problem), the same situation exists. Analytic solutions--mainly linear programming models--have been devised by Bowman [5], Manne [31], Wagner [47] and others, but at present they are computationally impractical and mainly of academic interest. Giffler and Thompson [16] describe an algorithm which generates from the set of all possible schedules a subset containing the optimal schedules. Computational effort is thus reduced, and in problems of small size the optimal schedule can be found. A heuristic approach to job-shop scheduling has been explored recently by Gere [15]. Programmed for a computer, it can handle reasonably large problems.

PERT, CPM, and Related Techniques

There has been a great deal written on the subject of large project scheduling since the fairly recent development of PERT (Project Evaluation and Review Technique) [30, 33, 50, 51], CPM (Critical Path Method) [21, 22, 23], and the flood of similar techniques which have followed. All are based on arrow diagrams, which show the jobs or activities of a project and their technological relationships. It will be useful for us to briefly review a number of these techniques, noting the approaches they take and their unique characteristics.

Originally designed for use on the Navy's Polaris missile research and development program, PERT was more of a planning and control technique than a scheduling tool, and it was essentially "time-oriented,", i.e., it paid little explicit attention to factors of cost and resource availability. Its most recent version--PERT/COST [51, 54]--is the result of government efforts to unify the many variations of PERT developed by the armed services and various businesses for use on weapon systems development projects contracted by the government (e.g., PERT II, PERT III, PEP, PEPCO, Super PERT, etc.). Essentially, PERT/COST adds the consideration of resource costs to the schedule produced by the PERT/Time procedure (as the earlier version is now called). There is no attempt to use cost data in such a way as to optimize total project costs, except by "manual" job shifting: where project costs indicate the necessity for excessive overtime or hiring, "manpower smoothing is accomplished by rescheduling slack activities to periods when the skills are not required by critical activities" [51, p.4]. PERT/COST is an example of an "enumerative" cost model.[1] All costs are merely enumerated, to facilitate comparison of projected and actual costs as the project progresses, rather than being used as parameters in an analytical cost-minimizing model. In this and other respects, the systems of PERTCO [11, 49] and SCANS [18] are quite similar, although the latter program has a routine for distributing slack in order to balance manning levels.

A "time-cost option" and a "resource allocation supplement" to PERT/COST are described in the DOD/NASA Guide [51]. A similar approach is recorded by Alpert and Orkand of ORI [2]. Alternative

1 The classification used here was proposed by Operations Research Inc. [20]; see also Clarke [8].

time-cost trade offs are estimated for each activity, and the schedule is shortened by "crashing" jobs on the critical path that cost the least per day shaved off the schedule, without regard to resource availability. When the desired due date is achieved, then resource leveling may be required. Slack jobs are shifted off peak days, or additional resources are applied to such jobs in an attempt to move them off the peak days. Smoothing is performed, after the computer program has produced a schedule, at management's option. Thus the computer program seeks to minimize required resources; it does not schedule by allocating given, limited resources. The methods proposed apparently have not yet reached the stage of a working model. The USAF PERT COST System Description Manual [March 1963] states, "Development and implementation of the Time Cost Options and the Resource Allocations Supplement have been considered a subject for future study" [54, p. i].

The Critical Path Method, developed by Kelley and others [21, 22, 23], represents a second major approach to the large project scheduling problem developed in recent years. Similar in many respects to PERT, which also uses a network graph to detect the "critical path" of jobs in a project, CPM differs mainly in that (a) it is a deterministic system (job times are assumed to take place without variations in planned time) as opposed to PERT's probabilistic approach (in which three times are estimated for each job--pessimistic, most likely, and optimistic, with associated probabilities); (b) it focuses explicitly on job costs, and presents an algorithm for minimizing project costs given a fixed due date.

Thus CPM represents a "single parameter optimization model" [8, 50]. Kelley describes the mathematical model upon which CPM is based as a parametric linear program [22]. Its solution depends on the Ford-Fulkerson algorithm for finding maximal network flows [13]. Basic to CPM is the assumption that a time-cost trade off exists for every job, and that the time-cost function is monotonic decreasing and concave between a job's normal and "crash" durations. (A linear relationship is usually assumed.) Although the model yields a minimum cost schedule for a project of fixed length, it assumes (as does PERT-Time) that resources are unlimited. The resulting schedule may be quite impractical in terms of available manpower or machines, and costs not included in the model (overtime, idle time, etc.) may be prohibitive.

Other related techniques are numerous. LESS [40] developed by IBM, uses essentially the Kelley approach. AMPERE [50] (an ORI system) calculates a schedule and cost data similar to PERT/COST, and then provides alternative courses of action management might take, utilizing normal, minimum and maximum loading for each skill group. PECOS [43], a more recent IBM program, combines some features of PERT and CPM: it still utilizes the probabilistic job times to calculate expected job times, and then seeks for the most efficient method of decreasing project duration by crashing critical jobs. Time-cost trade-off curves are assumed to be linear between normal and crash times. Based on an algorithm derived by Fulkerson [14], the program plots for a project an optimum time-cost curve, which is always piecewise linear under the assumption noted. As in CPM, the program assumes that resources are available as required.

McGee and Markarian [32] describe an "analytic technique" which they claim achieves an optimum allocation of manpower in a research/engineering project. It utilizes minimum and maximum manning levels for each job (activity) and assumes a linear relationship between the two in a time/manning plot. The method starts by assigning the minimum manning level to each job and calculating a critical path network. If the scheduled allocation of manpower exceeds given limits on any day, an attempt is first made to delay slack jobs on peak days. This failing, additional men are added as indicated. If the projected completion time is greater than the required due date, additional men are added to the jobs which compress the work at minimum cost. As with CPM, however, costs of varying resources are not considered. Essentially, the procedure smooths only by shifting slack jobs; beyond that it adds men as needed on peak days, without regard to costs of increased resource levels, idle time, etc. It does not consider the possibility of extending the due date (at some penalty).

A similar approach is taken by IMPACT, a model developed by Lockheed [48]. Activities are first scheduled at minimum loadings, and then the resulting schedule is compressed by increasing the manpower loading of activities which produce the maximum shortening per increment of added cost. Manpower limits are never exceeded, however; the model stops loading an activity when its shop limit is reached, and looks to the next cheapest activity to crash. The model does not consider the possibility of adding more manpower through hiring or overtime.

The main shortcoming that we see in many of the above scheduling models is that they do not take into account limited resources. Cost optimization, where attempted, is based on a single cost parameter: the premium for "crashing" jobs. In the models where limited resources are recognized explicitly, there is generally no attempt to consider alternatives of increasing resources or using overtime along with their associated costs; nor do they allow for comparison of penalty costs for extending the due date vs. the premium costs of adjusting resource levels.

While there are interesting features in many of the models, none of them have all of the "essential characteristics" we established earlier for an operationally useful scheduling model. Such a model, we believe, should deal more adequately with the major constraints generally associated with large projects.

A recent and ambitious computer model for project scheduling that does explicitly consider the restraints described earlier is RAMPS (Resource Allocation and Multi Project Scheduling), developed by duPont and CEIR, Inc. [52, 53]. As RAMPS is the proprietary program of CEIR, its details are carefully guarded, but the general approach it takes to scheduling is described in literature available from CEIR. RAMPS uses the basic network notions of PERT and CPM, though job times are deterministic rather than probabilistic. The program is designed to handle several projects with different due dates, taking into account (a) three different resource utilization rates for each job, with corresponding job times and work efficiences, (b) resource teaming on jobs, (c) penalties for splitting jobs, (d) resource limits for all shops

by periods, (e) costs of normal time, idle time, overtime and subcontracting in each shop, (f) project delay penalties, and (g) various management objectives (which can be ordered in importance), such as "minimize idle resources," "give priority to critical jobs," "work on as many jobs simultaneously as possible," and several others. We shall have more to say about RAMPS after we describe out own scheduling models in Chapter 4, when we will be in a better position to compare the two approaches to project scheduling.

A Linear Programming Approach to Project Scheduling

We have yet to investigate another avenue to the problem: the use of analytic techniques. The mathematical tool which seems best suited to minimizing some cost criteria given the constraints of a project setting is linear programming. Not only are computer programs readily available for the simplex algorithm, but an L.P. formulation has the advantage of providing the rich managerial interpretations that are available from the duality and sensitivity analysis features of the method. Its application to the large project scheduling problem is worth our investigation.

Charnes and Cooper have described a network interpretation and a directed dubdual algorithm for critical path scheduling [6]. The arrow diagram is viewed as a flow network, in which the initial node has unit input and the final node unit output. Flow variables, x_{ij} (which are restricted to the integers 0 and 1), are assigned to each link (job). Constraints reflect Kirchoff node conditions for conservation of flows over the system, i.e., the flow into each node equals the flow out of it. The function to be maximized is the sum of all x_{ij}, each weighted by a

coefficient equal to the time span of the associated link. In the dual problem, the variables w_i can be interpreted as the (negative) early start times assigned to each node. The difference between the w_i values for the first and last nodes is minimized, subject to the constraints that the difference between the w_i values of two connected nodes must be equal to or greater than the time required for the link (job) connecting the nodes. The value of the functional, then, is the total project time (and hence the length of the critical path), and the critical jobs are identified in the direct problem by positive x_{ij} values.

Unfortunately, this formulation of the problem does not include such data as resources required by each job and limits on shop resources. Implicit in the network flow interpretation is the assumption that all jobs along the critical path start at their early start times. As we will see in Chapter 3, when resources are limited, there may not be a critical path as ordinarily defined (a path of zero-slack, technologically related jobs from start to finish). In effect, the coefficients in the functional of the direct problem (and thus the stipulation constants in the constraints of the dual) lose their meaning.

It is possible, however, to write an L.P. formulation of the project problem which takes account of resource constraints. Using an approach similar to Bowman's for the job shop problem [54], we may develop a model as follows:

Subscripts:

s	shop (resource group)	$s = 1, 2, \ldots, m$
d	day (or other time period)	$d = 1, 2, \ldots, z$
j	job	$j = 1, 2, \ldots, n$
p	immediate predecessor of j;	$p \in P_j$ = [all immediate predecessors of j]

Variables:

x_{jd} activity of job j on day d; constrained to the integer values 1 (if job j is active) or 0 (if job j is inactive).

Constants:

a_{sd} men available in shop s on day d

c_{sj} crew size; men of shop s required on job j

t_j time length of job j, in days

Constraints:

1) $0 \leq x_{jd} \leq 1$ (and by integer programming techniques, x is constrained to equal either 0 or 1 [see Gomory, 17])

2) Jobs will be performed:

$$\sum_{d=1}^{z} x_{jd} = t_j , \qquad j = 1, \ldots, n$$

3) Capacity of shops will not be exceeded:

$$\sum_{j=1}^{n} c_{sj} x_{jd} \leq a_{sd} , \qquad \begin{matrix} d = 1, \ldots, z \\ s = 1, \ldots, m \end{matrix}$$

4) No job will be started before its predecessors are completed:

$$t_p x_{jd} \leq \sum_{i=1}^{d-1} x_{pi} \qquad \begin{matrix} \text{all } p \in P_j \\ d = 1, \ldots, z \\ j = 1, \ldots, n \end{matrix}$$

5) No jobs will be split:

$$t_j x_{jd} - t_j x_{j(d+1)} + \sum_{i=d+2}^{z} x_{ji} \leq t_j ,$$

$$j = 1, \ldots, n$$
$$d = 1, \ldots, z$$

Objective Function:

$$\text{Minimize } 1 \sum_{j=1}^{n} x_{jk} + 4 \sum_{j=1}^{n} x_{j(k+1)} + 16 \sum_{j=1}^{n} x_{j(k+2)}$$

$$+ \ldots + R_z \sum_{j=1}^{n} x_{jz} , \quad \text{where } k \text{ is some number such}$$

that $0 < k < z$, and $R_z = 4 R_{(z-1)}$.

Thus, the model seeks to find the shortest schedule given fixed resource constraints. For simplicity, no allowance is made for premium cost resources, such as overtime, hiring, subcontracting, etc. Nor is "crashing" or "stretching" a job allowed.

Not shown above are the additional constraints necessary to assure integer solutions (either 0 or 1) for the x_{jd}. Even without these constraints, however, the problem is a formidable one in terms of sheer size. If we assume the simplest version, consisting of constraints 1 through 4 only, even a small project is beyond the capacity of any present computer to handle. As an example, a project with 55 jobs in 4 shops with a time span of 30 days has some 5275 equations and 1650 variables (not counting slack variables or the additional equations and variables necessary to assure an integer solution). If job splits are not allowed, the number of equations increases to about 6870. Many of these equations, or course, are redundant;

and many of the variables could be eliminated from the start by calculating the early start times for all jobs assuming unlimited resources. Then all $x_{jd} = 0$, $1 \leq d < ES_j$. Nevertheless, even a trimmed-down formulation would exceed the capacity of most computers. A large machine--such as the IBM 7090 with 32 K storage--can handle a maximum of about 1000 variables in an L.P. program, thus limiting the application of this model to rather simple, small projects. The use of L.P. and a 7090 for such problems would be somewhat akin to using a bulldozer to move a pebble.

Linear programming formulations other than the ones above could be devised, of course, but the same difficulty would be faced: the scheduling of even medium sized projects (200 to 500 jobs) is an enormous problem, especially if resource limitations and other commonly encountered constraints are considered.

As for enumerative techniques, which exhaustively search the space of all possible schedules, none have been proposed--no doubt for the same reason that makes the L.P. approach impractical. We thus come to the same conclusion reached by Tonge after his examination of analytical and exhaustive methods for solving the line balancing problem [45, p. 15], and his words are appropriate in our own case:

> An approach that concentrates effort on those parts of the problem which seem to require it, rather than indiscriminately spinning out and eliminating possibilities at all stages of the solution process, would seem to be, a priori, a more feasible problem-solving procedure.

Heuristic Programs

Many of the scheduling models discussed earlier in this chapter use so-called heuristic techniques for problem solving, and it is upon this general approach to the large project scheduling problem that we will focus our efforts in the remainder of this volume. We use the term "heuristic" to mean, as suggested by Newell and Simon [37], a device or "rule of thumb" that reduces search in problem solving activity (e.g., "schedule all jobs at early start and move slack jobs off peak days"). In a more formal way [38], they define a heuristic program as a program for some relevant problem domain that "has some problem-solving efficiency for that domain--is capable of solving at least some problems," in contrast to an algorithm which they define as a program that "will produce a solution of any problem in D [the problem domain] in a finite number of steps." They further note that "the terms 'algorithm' and 'heuristic' program' are not antonyms, but designate different properties a program may possess." Often we may be primarily concerned with a program's heuristic power--"its capacity to find solutions rapidly," (relative to other programs applied to the same problem domain), which is quite independent of the program's algorithmic properties. Thus the simplex method of linear programming is an algorithmic program we could use (theoretically) for solving the large project problem, but it has little heuristic power in this application.

Our desire, however, is to develop a program which possesses the latter property--i.e., the ability to rapidly generate solutions--rather than algorithmic characteristics. In essence, we are sacrificing a guaranteed optimum solution for

reduced problem-solving effort. Rather than trying to exhaustively search the space of possible schedules for the best one, we will use cues in the problem environment to narrow our search to a sub-space rich in good schedules--though we risk the chance of missing the optimum solution altogether.[1]

Before discussing the heuristic devices employed in our scheduling model, we turn first to a consideration of some of the properties of large project schedules in the case of limited resources.

---

1 For more extended discussions of the use of heuristics in problem solving, and resumes of heuristic programs in use, see Simon and Newell [41], Simon [42], Newell, Shaw, and Simon [36], Gere [15], and Tonge [45].

Chapter 3

SOME PROPERTIES OF SCHEDULES FOR LARGE PROJECTS WITH LIMITED RESOURCES

Most of the recent methods proposed for scheduling large projects make use of a project graph (e.g., the arrow diagram basic to PERT and the Critical Path Method; see [22, 26, 51, 52]). The project graph is useful both for keeping track of the technological ordering of jobs in a project and for determining the degree of flexibility (i.e., the job slack values) available to the scheduler of the jobs. Given a project graph, which displays the predecessor-successor relationship of jobs in a project, and the times necessary to complete each of the jobs, one can then calculate the "critical path" or the longest ordered sequence of jobs through the project graph. Each of the jobs on the path is said to be "critical" or slackless; to delay any one of them would delay the completion date of the project. Other jobs with positive slack can be delayed up to the amount of their slack without such an effect, thus giving the scheduler some freedom in assigning start dates for each of the jobs.

This notion of criticality assumes, however, that unlimited resources are available for assignment to the project jobs (or at least that sufficient resources are available for each job to be scheduled some time between its earliest and latest start dates[1]). In the more usual (and general) case

---

1 Earliest start is defined as the earliest date a job can begin, given a project start date and the technological and time constraints of its predecessors. Latest start is the latest date a job can begin, given the same constraints of its successors, without delaying the project completion date. The difference between these two is the job's total slack.

where resources are limited, the above concept of criticality loses its meaning. Some jobs on a critical path may have to be delayed because of insufficient resources. If this occurs, then there no longer exists a start-to-finish path of technologically connected, slackless jobs. Under certain circumstances, however, one can identify a "critical sequence" of jobs in a project. As in a critical path, jobs in a critical sequence have zero slack,[1] and the length of the sequence determines the minimum length of the project. Unlike a critical path, a critical sequence is determined not by just the technological ordering and the set of job times, but also by resource constraints; furthermore, it is also a function of a given feasible schedule.[2]

As we proceed in this chapter, we will develop in detail the concept of a critical sequence and discuss some of its implications for project scheduling. It will first be necessary for us to explore the structure and properties of a project schedule, classifying several kinds of schedules and defining some operations we will perform on schedules. We will then extend the concept of slack to the case of limited resources and discuss the relationship of slack to schedule-generating rules. With this necessary groundwork laid, we will be able to define rigorously the concept of a critical sequence and to set forth the conditions which must be met in a project schedule

---

1 By a new procedure for calculating slack, which we will develop later.

2 The concept of "critical sequence" is a generalization of the Thompson-Giffler [16] concept of an "active chain" of operations in the job-shop scheduling problem. Other similarities will be evident as we proceed. We will later consider more carefully the relationship of the job shop and multi-project scheduling problems.

for a critical sequence to exist. We will complete the chapter by explaining how these concepts enable us to relate the job-shop and large project problems.

The Schedule and the Schedule Chart

We start with a project X characterized in the following manner:

1) The project consists of n separate, clearly-identifiable jobs or activities.

2) Associated with each job j is a time t required to complete the job,[1] a crew size cs, and a shop (or skill group, or machine group) s. If job j requires y different resources, then cs and s become y-dimensional vectors associated with j.

3) Also associated with each job j is a set P_j of jobs which are immediate predecessors of j, or jobs that must be completed before j begins. (P_j may be empty, in which case j is a starter job.) From the predecessor sets of all j's, we can infer for each j a set S_j of immediate successors. (S_j may be empty, in which case j is a final job.) A list of all P_j's (or of all S_j's) defines an ordering or technological relationship on the set of all jobs in X. We will use the symbol << to represent the relation "is an immediate predecessor of."

4) The jobs in X will be performed in m resource groups (shops, skill groups, machines, etc.), each containing a limited amount, a, of homogenous resources. Resource limits may vary from day to day, as men or machines are added or

---

1 We earlier noted our certainty assumption regarding job times. If PERT-type, three-point estimates are available (or any other distribution of job times), then E(t), the expected value for t, may be substituted for t in this analysis. See p. 86.

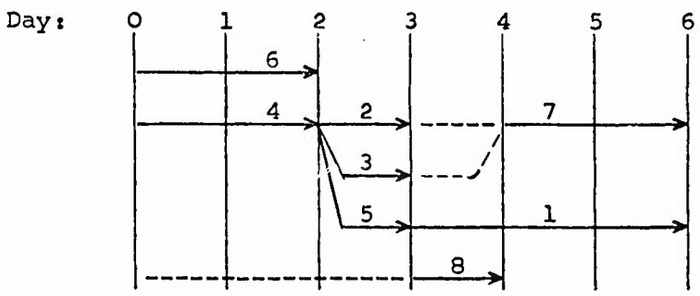
removed. We will henceforth refer to resources as men and a resource group as shop s. Thus the number of men available in shop s on day d is a_{sd}.[1]

Since "critical sequence" is a function of a project schedule, we must define more precisely what we mean by a schedule.

Definition: A schedule is a set of start times (AS) assigned to the jobs in X (one start time for each job). The length of a schedule z is the difference between the earliest of the start times of all jobs and the latest of the finish times of all jobs.[2] We will hereafter assume that the project start date S = 0; then the finish date F = z.

For purposes of clarity and simplicity in illustrating a project schedule, we will throughout this chapter make use of what we will call a Schedule Chart. It is in reality a combination of a traditional Gantt chart, which displays jobs scheduled along a horizontal time scale, and a project graph, which shows the technological ordering of jobs. The following is an example of a Schedule Chart.

Figure 3



1 If probability distributions for resource limits are available, then $E(a_{sd})$ may be used in place of a_{sd}.

2 Jobs, once started, are completed without interruption. Split jobs are considered as separate jobs, each with its own AS.

The time scale begins with day 0 on the left; each vertical line marks the end of one day and the beginning of the next. Jobs are shown by solid lines with an arrow marking the completion of the job. The horizontal span of a job represents its time length in days. Above each job is a number which represents its resource requirements, i.e., the crew size (cs) needed for the job. For convenience we will sometimes refer to the job by this number, e.g., "the 6-job." (In other cases where this would be ambiguous, we will further identify the jobs. If the jobs are performed in a number of shops, we could also label each job arrow with a shop number, or with several shop numbers if the job requires multiple resources.) Technological orderings are shown by connecting the jobs, either directly or by dotted lines. Thus the above chart summarizes the following information:

Job Number (same as Crew Size)	Length (days)	Predecessors	Start Date	Finish Date
6	2	-	0	2
4	2	-	0	2
2	1	4	2	3
3	1	4	2	3
5	1	4	2	3
8	1	-	3	4
1	3	5	3	6
7	2	2, 3	4	6

Associated with a given Schedule Chart we could draw, for each resource group employed, a Resource Requirements vector $Q_s = (q_1\ q_2\ \ldots\ q_z)$, showing the total resources of shop s required on each day d of the schedule. For example, if all jobs in the above project occur in the same shop s, the resource requirements vector for the schedule would be Q_s = (10 10 10 9 8 8). In a similar manner, we could draw a Resource Availability vector $A_s = (a_1\ a_2\ \ldots\ a_z)$, showing the

resources available in shop s on each day d. Note that A_s may be considered a row vector in an m x z matrix A whose entries are a_{sd} as defined earlier. Similarly, Q_s is a row vector in a resource requirements matrix Q having the same shape, with q_{sd} analogously defined.

For the present we will be concerned with the special case where shop limits are constant over time (i.e., $a_{sd} = a_{s(d+1)}$, $1 \leq d < z$). Later we will relax this restriction and discuss the more general case where shop resources may vary over the schedule period.

The Schedule Chart now assists us in classifying different types of schedules, and in defining the operation of "job shifting."

Definition: A <u>feasible schedule</u> is a schedule for which a Schedule Chart can be drawn and for which $Q \leq A$. That is,

1) the technological ordering and job times are observed (i.e., no job is scheduled until all of its predecessors have been completed);

2) the resource constraints are not violated (i.e., the number of men scheduled never exceeds the number available; and

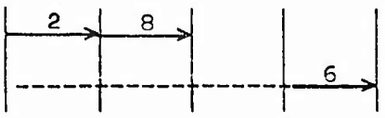
3) the length of the schedule is finite.

Definition: An <u>optimum schedule</u> is a feasible schedule whose length is at least as short as the length of any other feasible schedule.

Definition: Consider the Schedule Chart of a feasible schedule and a set of shop limits A. Pick any job j starting on day d and reschedule it to begin on day d-1 (keeping within the bounds of the original start and stop dates of the project). If j occurs in shop s, calculate the new manpower loading for shop s on day d-1. If this is $<q_{sd}$, then the schedule is still feasible

and we say that we have left shifted job j by one day.[1] Likewise if j can be delayed to begin on d+1 without delaying the project finish date and without q_{sd} being exceeded on day d+t where t is the length of job j), then j can be right shifted one day. A left shift of i days ($i \geq 1$) is a local left shift if it can be accomplished by a series of one-day left shifts, each of which maintains the feasibility of the schedule. A local right shift of i days is defined analogously. A global left shift is a left shift of any job j that results in a feasible schedule which could not be obtained by local left shifting of j. (Thus a global left shift is always a shift of more than one day.) A global right shift is defined analogously.

Consider the following example:



The 6-job can be locally left shifted one day. If it were left shifted 3 days, the resulting schedule would be feasible, but the left shifting would be global rather than local.

Definition: A left-justified schedule is a feasible schedule in which, because of technological orderings and/or resource constraints, no job can be started at an earlier date by local left shifting of that job alone. (Figure 3 above is left justified if the shop has a resource limit of 10.) A right-justified schedule is analogously defined.

Definition: An associated right-justified schedule is a right-justified schedule that can be derived from a given left-justified schedule by a series of local right shifts. An associated left-justified schedule is similarly derived from a right-justified schedule by local left shifting.

---

1 If j is a multi-resource job, the new manpower loading for each relevant shop must be calculated and compared with q_{sd} of that shop.

Slack

In the case where resources are not limiting, the notion of slack is simple and unambiguous; there is a single slack value associated with each job. This derives from the definition of slack[1] and from the fact that there is a unique left-justified schedule and a unique right-justified schedule for a given project. Thus the early start (ES) and late start (LS) values for each job are independent of the order in which jobs are scheduled (technological restraints being observed, of course).

Such is not always the case when resources are limited. There may be several right and left-justified schedules for every project. In general, for each project and set of shop limits there is a non-empty set J_l of left-justified schedules and a non-empty set J_r of right-justified schedules. And for each schedule x in J_l, there are one or more associated right-justified schedules which comprise a proper subset of J_r.

Consider, for example, the simple project that follows, in which all jobs are performed in the same shop:

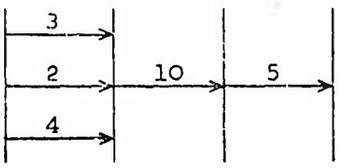
Job Number (same as Crew Size)	Predecessors	Length (days)
3	-	1
2	-	1
4	-	1
10	2	1
5	10	1

With unlimited resources, the project would have the following schedules:

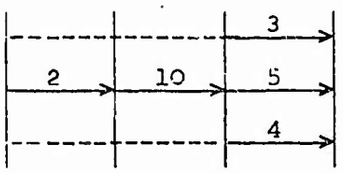
---

1 We are here talking about total slack, which is defined as the difference between the late start and early start of a job. Other types of slack have also been defined, e.g., free slack, the difference between a job's early finish and the earliest start of any of its successors. See Kelley [22].

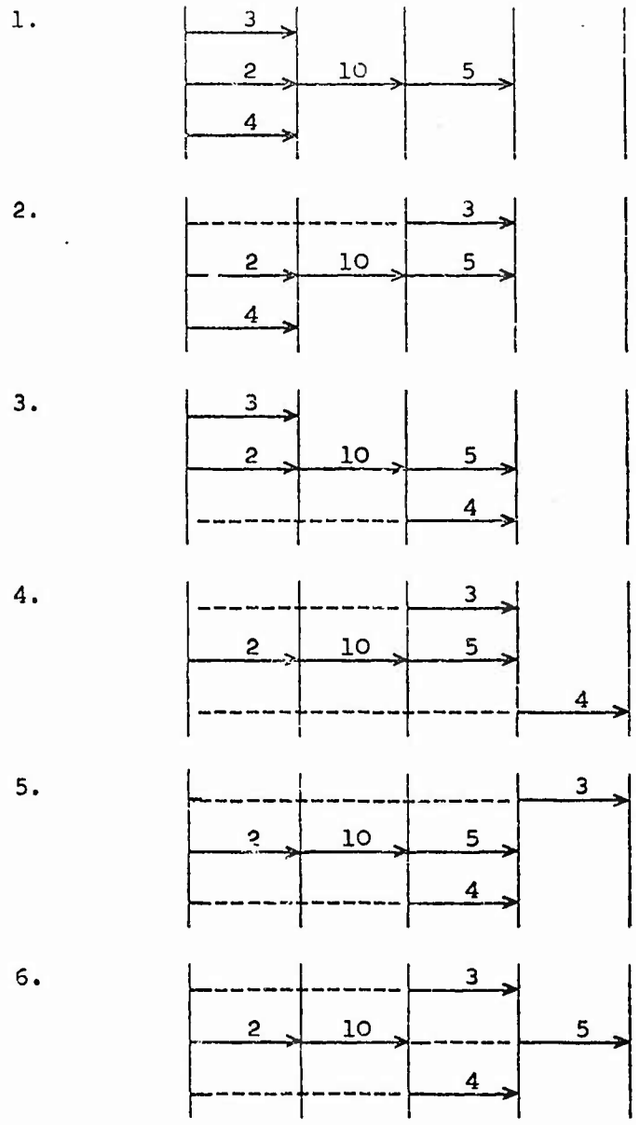
Left-justified:



Right-justified:

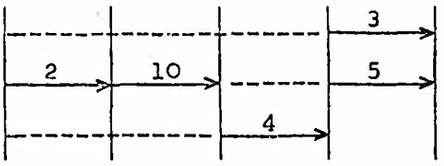


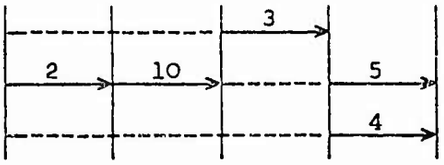
Jobs 3 and 4 would each have slack of two days and the other jobs, being critical, would have zero slack. If the shop has a resource limit of 10, however, then J_1 contains six different schedules:



If we allow local shifting only, schedules 1, 2 and 3 are also right justified, since no jobs can be right shifted; then all jobs in these three schedules are "critical"--i.e., they have zero slack. (Note that in schedule 1, a global right shift of job 3 past job 10 would result in a feasible right and left-justified schedule--i.e., schedule 2.) Schedules 4, 5 and 6

each have two associated right-justified schedules. For example, schedule 6 has the following associated right-justified schedules:





Thus the traditional notion of slack is ambiguous. For three of the jobs (3, 4, 5) ES and LS depend on the particular schedules we choose, and thus no single values for slack (in the usual sense) exist for these jobs.

Obviously, then, the ordinary methods of calculating slack do not suffice in the limited resource case. If we retain the idea that slack represents the amount of time a job can be delayed from its early start without delaying the project completion date, then we must recognize the conditional nature of slack when resources are limited. Slack values are related to a given pair or right and left-active schedules and are thus conditional upon the rules or procedures for generating these schedules. While this notion adds some complexity to the simple slack calculation of the unlimited resource case, it still preserves much of the operational utility of the slack concept. And by a judicious choice of schedule-generating rules, one may retain, if he desires, some of the useful characteristics that slack has in the unlimited resource case. For example, in

the latter case:

1) Slack is easily and unambiguously calculated: a unique set of slack values are associated with a given project; and

2) Slack is continuous (convex) over its range--i.e., a job with k days of slack may be delayed anywhere from 0 to k days without delaying non-successor jobs of the project finish date (thus giving the scheduler or shop foreman some flexibility in assigning job start times).

We could easily preserve characteristic 1) above, in the limited resource case, by devising a set of rules or procedures for generating a single left-justified schedule and a single right-justified schedule. Then a unique set of slack values could easily and unambiguously be calculated from the two schedules. Further, if we chose our rules in such a way that the right-active schedule was associated with the left-active schedule--i.e., derived from the latter by local right shifts only--then characteristic 2) would also be maintained.

It is obvious, however, that we could devise many different scheduling rules or procedures that would result in quite different schedules, and there is no a priori way of deciding which rules are best. One set of rules may result in a "fortunate" assignment of slack values for one project (e.g., slack values that enhance the possibility of smoothing the schedule through juggling the slack jobs) but may work less satisfactorily than another set of rules when applied to a second project. (We will later see an example of this.) And situations can easily be imagined where global shifts, if permitted, might be operationally preferable to local shifts only, even though the convex property of slack might be lost. For

example, a foreman might find his work scheduling easier if he were told that job j could be delayed either exactly 7 days or between 1 and 3 days. If convex slack only were allowed, the possibility of a 7-day delay would not be discovered.

For large and complex projects, one would have difficulty even enumerating all the elements of J_r and J_l with their associated conditional slack values. However, one could fairly easily devise several sets of rules that represent reasonable alternative approaches for generating schedules, apply each of them to the project, and compare the resulting slack values, using some measure or criteria of suitability of results (e.g., do the slack values obtained permit juggling which results in more efficient use of resources?). Another approach would be to apply the rules probabilistically. One could keep track of slack values thus generated to obtain the bounds on slack given certain resource limitations. The slack (TS) for any job j satisfies

$$TS_j \leq \max_{y \in J_r} LS_j(y) - \min_{x \in J_l} ES_j(x)$$

where x and y are feasible schedules of the project. The upper bound on TS_j is the slack value for j given unlimited resources.

Whatever the method used to calculate slack, it should be clear that any set of slack values is based on a given pair of right and left-active schedules and hence is <u>conditional</u> on the schedule-generating rules.

<u>Schedule-Generating Rules</u>

The problem of developing "reasonable" rules or procedures for generating left and right-active schedules deserves come comment here. Obviously, all left-justified schedules for a given project are not equally "good," in general, if we assume some

measure of "goodness" such as minimum schedule length or maximum utilization of resources. Using these criteria, for example, we would conclude that schedule 1 in the above project is better than schedules 4, 5 or 6; it is shorter and has a higher daily average utilization of resources.

As we noted in Chapter 2, only an analytic solution would guarantee us an optimum schedule (as defined above); the heuristic methods aim at a satisfactory solution with more reasonable computational effort. Most of the network-based methods for project scheduling discussed in Chapter 2 would probably generate a left-justified schedule, however, since they either assume unlimited resources and schedule all jobs at early start (in which case slack calculations are unambiguous), or they attempt to schedule all jobs at early start, delaying those that occur on peak days only enough to reduce the peak loads to some desired level. However, our concern here is not how to generate a left-justified schedule, but rather, how to calculate job slack values for some given schedule.

If we assume, then,that a left-justified feasible schedule has been generated by some means, what would be a reasonable approach to determining a set of slack values for the jobs? This amounts, of course, to defining a procedure for obtaining a related right-justified schedule. As we previously noted, several different right-justified schedules may be related to a given left-justified schedule. The order in which jobs are right shifted accounts for these differences. As an example of a procedure that might be used, we describe below some rules that generate a right-active schedule from a given left-active schedule by a series of local right shifts. The rules are simple and unambiguous, and have several other virtues that

will become apparent.

1) Given a left-justified schedule, select from it the set of jobs whose finish time (EF)[1] is a maximum (there will be one or more jobs in this set; their EF determines the project finish date F). For each of these jobs, set LS = ES, LF = EF, and TS = 0. Set i = 1.

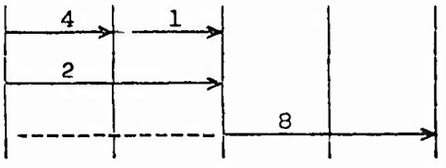
2) Consider now the jobs whose EF = F - i. If the set is empty, go on to the next step. If the set contains exactly one member, right shift the job until it meets a technological or resource constraint of jobs already considered in previous steps (or, if none exist, until the job's finish date = F). At this point, set LS = ES + the number of days the job was right shifted, LF = LS + t, and TS = LS - ES. If there is more than one job in the set, the priority of right shifting is determined as follows: calculate for each job separately an LS (assuming in each case that no other jobs in the set have been right shifted). Arrange the jobs in descending order of their LS (i.e., the latest LS first) and right shift the jobs in that order. In case of a tie in LS's, arrange the tied jobs in ascending order of manpower requirements (i.e., the smallest crew-size jobs first) and right shift the jobs in that order. If there are still ties, then decide on the order of shifting by random selection from the tied jobs. As each job is right shifted, calculate its LS, LF, and TS as above.

3) Set i = i + 1. If i < F, then return to step 2). If i = F, then stop; all jobs have been considered, and a unique slack value assigned to each of them

---

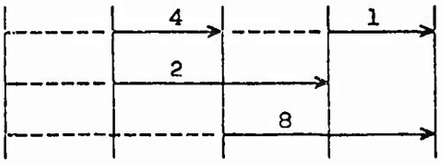
1 The following notation is used throughout: ES (early start), EF (early finish), LS (late start), LF (late finish), TS (total slack).

The main advantage of these priority rules is that they tend to move first the shortest jobs and/or the jobs that can be right shifted the furthest. In general this will tend to distribute potential slack to the largest number of jobs, rather than distributing larger amounts of slack to fewer jobs. (We will see later that local suboptimalities in a schedule can be removed if all jobs in a local group have slack.) Consider the following example:



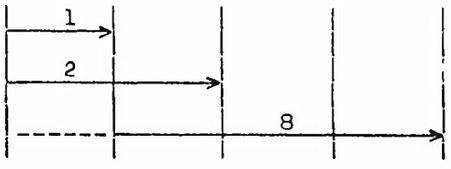
(shop limit = 10)

In this left-justified schedule, if the 2-job is right shifted first, then the 4 and 1-jobs will have no slack. But if the above priority rules are followed, the 1-job will be right shifted first (2 days), then the 2-job (1 day) and the 4-job (1 day). Hence all three end up with slack. The resulting right-justified schedule (below) could obviously be shortened if all jobs were left shifted one day.



(shop limit = 10)

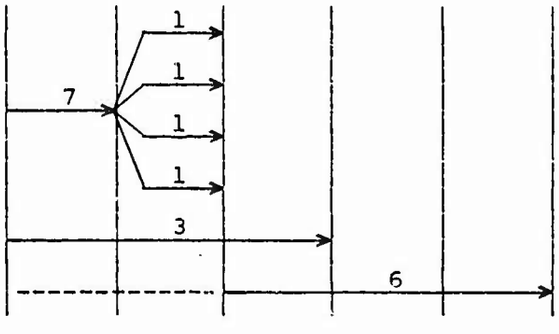
It should be noted that right shifting jobs in decreasing order of their EF has the advantage of sometimes removing resource "bottlenecks"--for example:



(shop limit = 10)

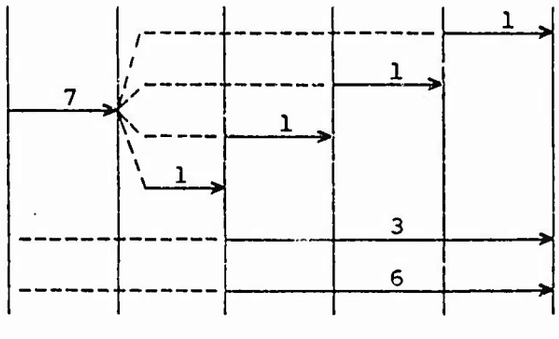
The 2 job forms a bottleneck past which the 1-job cannot be moved. (This assumes, of course, only local right shifting is permitted, so that slack values obtained are convex. Rules allowing global right shifts would ignore such bottlenecks and might have some advantages, if slack convexity is not required.) The 1-job can be right shifted, however, if the 2-job is moved first (as the priority rules dictate)--hence the advantage of shifting jobs as they are encountered, going from right to left across the Schedule Chart.

The priority rules above will not bring such favorable results with all schedules, however. For example, the schedule



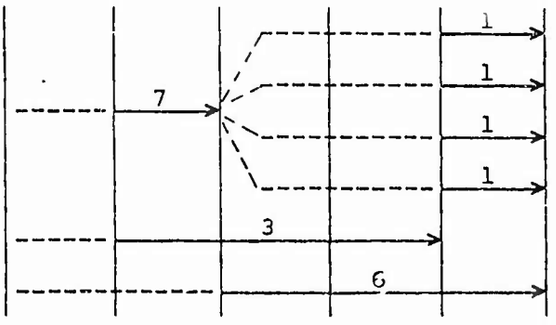
(shop limit = 10)

would be right shifted as follows according to the priority rules:



(shop limit = 10)

One of the 1-jobs ends up with no slack. Had all of the 1-jobs been right shifted before the 3-job, however, the schedule would appear thus:



and all but the 6-job would have slack. The schedule could obviously be shortened. A different set of priority rules would be needed to discover such situations. As suggested above, some combination of priority rules, perhaps applied probabilistically, might lead to the best overall results. Another approach would be to construct, alternately, a right-justified schedule, then a left-justified schedule, another right-justified schedule, and so forth, back and forth, each one derived from (i.e., associated with) the previous schedule, with the hope of finding larger slack values. This procedure could be continually repeated until one approaches the maximum slack values for each job (or an "optimum" distribution of slack, according to some criteria).

We have now accomplished our immediate objective of extending the concept of job slack to the limited resource case, noting its reliance upon a set of schedule-generating priority rules and giving an example of such a set of rules. Formally, we may define slack, in either the limited or unlimited case, as follows:

Definition: $TS_j(x,y) = LS_j(y) - ES_j(x)$, $x \in J_l$, $y \in J_r$. That is, the total slack of job j, relative to schedules x and y, equals the late start of j in schedule y minus the early start of j in schedule x. If x and y are associated with each other, the slack values are convex. If resources are not limiting, then J_l and J_r each contain just one schedule and the slack values are unique.

We are now prepared to discuss the concept of critical sequence.

The Critical Sequence

In the case of unlimited resources, the string of critical jobs which determines the minimum project length is aptly named the "critical path." On the project graph it can be traced as an unbroken sequence of technologically ordered jobs; it forms a literal path from start to finish. The analogous concept we are developing for the limited resource case differs in that a technologically connected path of critical jobs does not always (in fact, does not usually) exist; but a sequence of critical jobs can nevertheless be identified--hence the term "critical sequence." Further, it is composed of one or both of two types of sequences which we shall now define:

Definition: Given a project X, a technological sequence of jobs is a set T of two or more jobs technologically connected in a linear (non-branching) sequence (i.e., the i^{th} job in the sequence is the immediate predecessor of the $(i+1)^{th}$ job). In a given feasible schedule, a technological sequence is joint if there are no intervening time periods between the completion time of one job and the start time of its immediate successor (when both are in the sequence). That is,

$$AS_j + t_j = AS_k, \qquad j \ll k, \qquad j \text{ and } k \in T.$$

If $AS_j + t_j < AS_k$ for any two jobs j and k ε T, j << k, the sequence is disjoint at their juncture. A joint technological sequence is in essence a local critical path; to delay any job in the sequence would delay the completion of the whole sequence.

Definition: Given a feasible schedule, a <u>resource sequence</u> of jobs is a set R_s of two or more jobs that require the resources of the same shop s and that do not overlap in time. (A job which requires multiple resources may be a member of several resource sequences.) A resource sequence is <u>joint</u> if the jobs in the sequence span an interval of time Z with no overlaps or gaps. That is, if the jobs are arranged in increasing order of AS, then

$$AS_j + t_j = AS_k \text{ for any two adjacent jobs j and k}$$

in the ordered list, and $\sum_{i \in R_s} t_i = Z$. Such a sequence of jobs may or may not be technologically related.

We are now able to define critical sequence.

Definition: In a given feasible schedule, a <u>critical sequence</u> (if it exists) is a set CS of one or more jobs that has the following properties:

1) All jobs in CS have zero slack.[1]
2) If CS contains more than one job and if the jobs are arranged in ascending order of AS, then
   a) any two adjacent jobs in the list are co-members of either a joint technological sequence or a joint resource sequence (or both);
   b) the first job in the list is also the first member of every joint technological and/or resource sequence to which it belongs; and
   c) the last job in the list is also the last member of every joint technological and/or resource sequence to which it belongs.

---

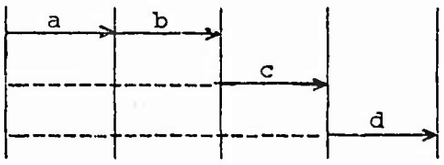
1 As calculated by the procedure outlined above, or some other schedule generating procedure.

As a consequence of these properties and previous definitions, we can observe the following features of a critical sequence:

1) The length of the critical sequence (and hence the project) is

$$z = \sum_{i \in CS} t_i$$

2) In more descriptive terms, a critical sequence (of two or more jobs) follows either a joint technological sequence or a joint resource sequence, or an alternating combination of both.[1] Note that, as a consequence of 2) - a) above, when a shift from one to the other occurs, the two sequences share at their juncture a job which is common to both. Thus if a given technological sequence is followed by a resource sequence, the last job in the former occurs in the shop of the latter and is therefore also a member of the latter sequence. To illustrate, consider the following portion of a critical sequence.

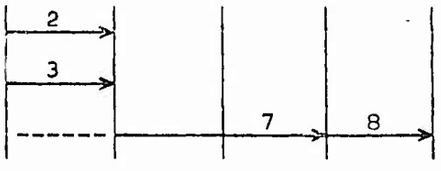


Assume job a is in shop 1 and jobs b, c, and d are in shop 2. As the schedule graph indicates, a << b and a and b form a joint technological sequence. But b is also a member of a joint resource sequence, b - c - d. The same sharing condition holds when a resource sequence is followed by a technological sequence; the first member of the latter is also a member of the former.

---

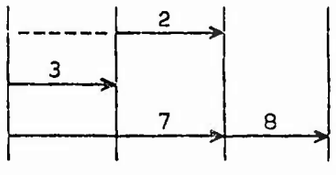
1 Since a multi-resource job belongs to several resource sequences, the one of relevance here involves the resource which constrains the job from left or right shifting.

Not every left-justified schedule will have a critical sequence of zero slack jobs. Consider the following left-justified schedule:



(shop limit = 10)

Both the 2 and 3 jobs have unconditional slack (by our established procedure, 2 would have three days and 3 would have two days slack). A critical sequence, therefore, does not exist. However, the schedule can easily be shortened as follows:



And in this schedule, a critical sequence does exist (consisting of the 7 and 8 jobs). The first schedule exhibits what we will call a local suboptimality. We will see that local suboptimalities may be removed easily, and that a left-justified schedule with no local suboptimalities always has a critical sequence. We need first to define some terms.

Definition: Local Set $L = [l_1, l_2, \ldots l_n]$: Given a left justified schedule, a local set is any set of one or more jobs which (a) have the same early start (we will refer to it as ES_L), (b) are in the same shop, and (c) are resource constrained only. (If the set contains one job only, requirements (a) and (b), of course, are superfluous.)

Definition: Constraining Set of a local set $G = [g_1, g_2, \ldots g_n]$: Given a local set in a left-justified schedule, a constraining set consists of those jobs which constrain the local set from further left shifting, i.e., all jobs using the same resource as L such that $EF_G = ES_L$.

Definition: Concurrent Set of a local set $C = [c_1, c_2, \ldots, c_n]$: Given a local set in a left-justified schedule, a concurrent set consists of those jobs which are concurrent to the beginning day of the local set (but not members of that set); i.e., all jobs c such that $ES_c \leq ES_L < EF_c$, $c \notin L$.

Definition: Local Suboptimality: A left-justified schedule is said to contain a local suboptimality if it contains a local set for which all jobs in the constraining set and concurrent set have slack.

Definition: Left-Active Schedule: A left-justified schedule will be called a left-active schedule if it contains (a) no local suboptimalities and (b) no jobs which can be started earlier by global left shifting.[1]

Although all five definitions above apply to a left-justified schedule, they are symmetrical in the left-right dimension. Thus the definition of a right-active schedule follows from the above definitions by changing "left" to "right," "earlier" to "later," "beginning" to "ending," "ES" to "LF," and "EF" to "LS."

With the aid of the above definitions, we may now formally prove the existence of a critical sequence in a left-active schedule. (The following theorems and corollary assume

---

1 We should note here the similarity of "critical sequence" to the Giffler-Thompson notion of an "active chain" in the job-shop problem [16, p. 493]. Likewise, our "left-active schedule" is analogous to their "active schedule," which they define as "a feasible schedule having the property that no operation can be made to start sooner by permissable left shifting." Our additional requirement of no "local suboptimalities," (noted above for a left-active schedule) reflects the more complex nature of the project scheduling problem. In the latter case, a job (operation) may be scheduled on any one of several identical facilities (e.g., machines) that are available in a given shop. (In the Giffler-Thompson case, each facility is unique.) Thus it is possible for a schedule to be "active" in the sense that no jobs can be left shifted (locally or globally) and still not contain a critical sequence, as noted above (p. 51). The Giffler-Thompson job-shop problem, then, may be considered a special case of the large project problem.

that resource limits are constant over the schedule period.)

Theorem 1: A schedule can always be shortened if it contains a local suboptimality.

Proof: It is obvious that if all jobs in a Schedule Chart that occur on a given day[1] are left shifted one day, then the project finish date may be reduced by one day, for all succeeding jobs may then also be left shifted one day. Thus if we can prove that a local set and all its concurrent jobs can be left shifted one day, we have proved the theorem. Begin by right shifting[2] all the concurrent jobs and constraining predecessors of the local set. This is possible since, by the definition of local suboptimality, all have slack. The new feasible schedule obtained has, of course, the same finish date as the old. Now left shift by one day the local set (L) plus all of the jobs presently concurrent with it (C). (Some of the jobs previously right shifted--including all jobs globally right shifted--may have moved beyond the point of concurrence with L). The legality of this move may be examined by considering the two possible types of constraints: (a) technological and (b) resource constraints. Since we have assumed the level of resources in all shops is constant over the scheduling period, then technological and resource constraint points to left shifting occur only at the tail-end of jobs on the Schedule Chart, i.e., at their EF.

Consider constraints of type (a): Note that the jobs in C originally had $ES < ES_L$, or else they would not be concurrent with L after the right shifting. And since no jobs with $EF < ES_L$ were right shifted, then none of the

---

1 This includes both jobs that start on the given day and jobs started earlier, but still active (unfinished) on that day.

2 The order of shifting outlined in whatever procedure was used for calculating slack values may be followed, or any other order that will allow all jobs involved to be locally or globally moved at least one day.

predecessors of C were moved. Hence there are no technological constraints to a one-day left shift of jobs in C. Nor are there like constraints for L, since jobs in this set were resource constrained only.

Now consider type (b) constraints. Note that all jobs originally right shifted had $EF \geq ES_L$. Thus, that shifting could not have caused any of the constraints to the left of ES_L to become more severe; i.e., right shifting of those jobs could not have increased the resource load on any day prior to ES_L, in any shop. And since jobs in C are to be left shifted no more than they were right shifted, then the left shifting cannot increase resource loads on days prior to ES_L above their original levels.[1] Therefore no new resource constraints are created by left shifting and the jobs may be moved the one day required. The proof is complete.

Note that the resulting schedule is not necessarily left justified, and local suboptimalities may still exist. Additional left shifting and continued application of the above procedure will lead eventually to the removal of all suboptimalities. We may state this as a corollary:

Corollary: Any left-justified schedule may be converted into a left-active schedule by continued right and left shifting.

Proof: If the left-justified schedule has no suboptimalities and no possibilities for global left shifting, it is already left active. If global left shifts are possible, perform them. If the schedule contains a suboptimality, then shorten the schedule by the above procedure and left justify the result.

---

1 Except for the day just prior to ES_L which may be increased by left shifting of the local set; however, as we have noted, this creates no problems as there are no jobs with $EF = ES_L$ and hence no constraints to left shifting on that day.

If it still contains a suboptimality or possible global left shifts, repeat the process. By a series of such moves (finite in number) all suboptimalities may eventually be removed and the schedule will be left-active.

Theorem 2: Every left-active schedule contains at least one critical sequence.

Proof: The proof is based on the observation that no job j can have zero slack unless one or more of its constraining successors also has zero slack; for if all such jobs could be right shifted, then so could j. Thus, if we can identify a zero-slack job anywhere in the left-active schedule, we can trace a joint sequence of zero slack jobs from that point in the schedule to its termination. More specifically, we shall prove that at least one of the jobs which begin on day zero has zero slack, and therefore, by the above reasoning, a critical sequence of zero slack jobs exists from start to finish.

The proof will be presented in the form of an algorithm:

(A) Consider the set J of all jobs having EF = F. All have TS = 0.
Pick one of these jobs and call it j.
If $ES_j = 0$, go to (E).
Since $ES_j > 0$, then there must exist a left-constraining set of j.

(B) Examine G_j (the constraining set of j).
If $TS_g > 0$ for all $g \in G$, go to (C).
Pick a job g for which $TS_g = 0$; call it j.
If $ES_j = 0$, go to (E).
Since $ES_j > 0$, then j must have a constraining set. Go to (B).

(C) Examine C_j (the concurrent set of j).
If $TS_c > 0$ for all $c \in C_j$, go to (D).
Pick a job c for which $TS_c = 0$; call it j.
If $ES_j = 0$, go to (E).
Since $ES_j > 0$, then j must have a constraining set. Go to (B).

(D) A local suboptimality exists, contrary to the hypothesis of a left-active schedule. Therefore, both $TS_g > 0$, all $g \in G_j$, and $TS_c > 0$, all $c \in C_j$, cannot hold.

(E) We have found a job (j) which begins on day 0 and has zero slack. Therefore, a critical sequence exists and the proof is complete.

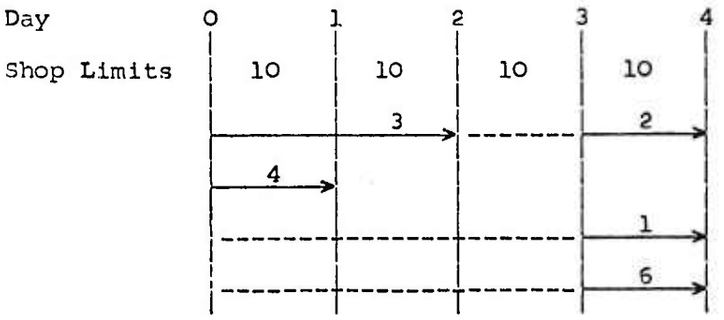
Note that each time the search cycles through step (B), the job being examined has an earlier ES than the previous job considered, since all jobs in either G_j or C_j have $ES < ES_j$. Thus one is assured of eventually reaching a job whose ES = 0, since the schedule is finite in length.

Implications of the Critical Sequence Concept for Project Scheduling

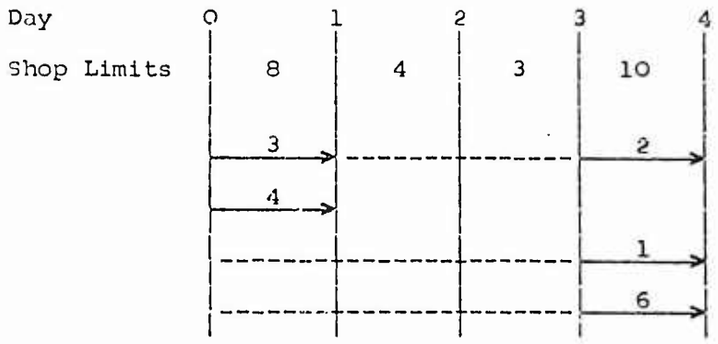
Identification of a critical sequence has much of the same utility, for purposes of scheduling, as the designation of a critical path in the unlimited resource case. In order to shorten a schedule, for example, only jobs on the critical sequence need to be considered. Where critical sequences exist in parallel, all must be contracted to have any effect on the project finish date. Critical jobs may be shortened by improved technology or by additional assignment of resources ("crashing" the job). The latter is possible, of course, only if resource limits have not been reached on "crash" days (or if overtime or subcontracting is allowed). The Kelley approach [22] to minimizing project costs by "crashing" critical jobs (until the costs of doing so exceed the savings that result) may also be used in the limited-resource problem, if "crash" costs are modified when necessary to include costs of overtime and subcontracting.

Variable Resource Limits

We have thus far considered the special case of resource availability where resource levels are constant throughout the scheduling period. In this case constraint points to left shifting occur only at the project start date and at terminal points (AF) of already scheduled jobs; likewise, constraints to right shifting occur only at the project finish date and at the beginning points (AS) of already scheduled jobs. However, if the available resources are uneven over the scheduling period (i.e., if $a_d \neq a_{d+1}$ for any $1 \leq d < z$, where the a's are entries in the resource availability vector A_s), then constraint points can occur on days when resources change, whether or not jobs end or start on the same days. To illustrate:

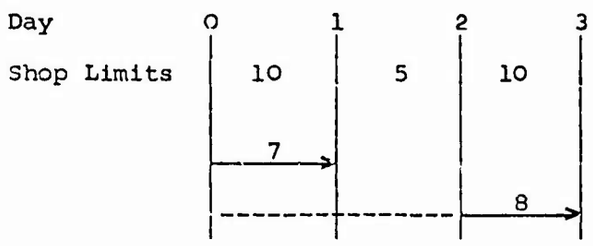


Obviously, job 2 could be left shifted to start at day 2, job 1 to day 0, and job 6 to day 1. In every case, the constraining point is either the terminal point of another job or the start date of the project. Consider an example where resources are uneven, however.



Job 2 could be left shifted to start at day 1 and job 1 to day 0, but job 6 could not be moved at all--even though no job ends at day 3 to constrain it. Job 6 is effectively constrained by the reduced resources on day 2 (as compared to day 3). Thus, the previously described notion of a critical sequence does not readily apply here. The schedule cannot be shortened (if the original start date is observed), yet no critical sequence of jobs from start to finish exists.

In general, a critical sequence may or may not exist in a project with variable resources. If a schedule may be shortened from either end (start or finish), sometimes a critical sequence may be created (as is possible in the example above); but this is not always possible. For example:



Both jobs 7 and 8 have zero slack, but no critical sequence exists and the schedule may not be shortened given the resources available.

Theorem 3: If resources available are a decreasing function of time (i.e., $a_d \geq a_{d+1}$, $1 \leq d < z$), a critical sequence always exists.

Proof: Since $a_d \geq a_{d+1}$, there is no possibility of left shifts being constrained by reduced resources. Hence the only points of restraint are job terminal points, and the proof given for Theorem 2 applies.

Whether or not a complete critical sequence exists, there will always be in every project one or more jobs with zero slack (e.g., every job whose early finish in a left-active schedule equals the project finish date). The limited-resource concept of slack discussed above applies equally well in the variable resource case, and the operational utility of slack is unchanged. Conceptually, one could calculate slack values for jobs in a project (following some rules or procedures for generating right and left-active schedules) and attempt to minimize project costs using the methods discussed on p.56.

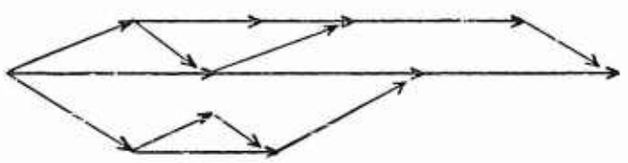
Relationship of Job-Shop and Large Project Problems

We have earlier noted the similarities between "active chain" in the job-shop problem and "critical sequences" in the large project problem. Here we will consider the relationship of the two problems in greater detail:

Typically a "job" in a job-shop consists of a sequence of operations to be performed in a given order on some object. Each operation requires a different facility (machine or other resource) for a given time period. Conceptually, a job can be pictured as a simple, single chain network:

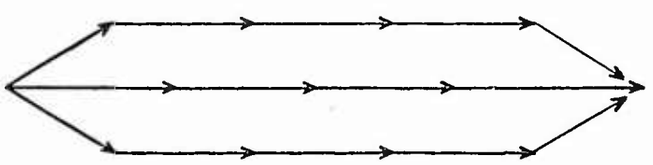
Each operation (except the beginning or ending one) has one predecessor and one successor operation. (In some cases, the exact order of operations is arbitrary, but only one operation is performed at a time.)

In contrast, the large project problem is typically characterized by a more complex technological ordering. Each job may have several predecessors or successors; jobs may be performed in series or parallel, as indicated by their partial ordering:



In the job-shop problem, the finish date of each job is of concern; in the large project problem, only the project completion date is important.

Note that a project may look like a number of job-shop chains connected at the start and finish:



And by use of dummy jobs (having given time lengths but zero resource requirements), the job-shop chains can be constrained to start no sooner, and finish no later, than any desired start and stop dates (which, of course, must be sufficiently separated to permit completion of all operations in a job chain). Thus the job-shop problem can be considered a special case of the large project problem. An analytic solution to the latter problem conceptually may be applied to the former. Likewise, a

given heuristic approach to large project scheduling should be applicable to job-shop scheduling problems, though its power in the latter case may differ from that in the former.

Chapter 4

COMPUTER MODELS FOR LARGE PROJECT SCHEDULING

As we have noted, those who follow a so-called heuristic approach to problem solving are interested in gaining "heuristic power" in their program--i.e., computational efficiency. They do this either because an algorithmic program is unavailable or because such a program is computationally infeasible.[1] In the latter case, they may "sacrifice" an optimum solution (if indeed the calculations could ever be completed) for the program's capacity to rapidly find a satisfactory solution.

Similarly, our goal has been to develop a combination of scheduling rules into a program which can quickly generate a good project schedule. This "heuristic power" of the program enables us to include probabilistic elements in the program rules (explained below), so that a number of different schedules can be generated in a reasonable length of time and the best one selected from the group. In this manner we hope to increase the probability of finding an optimum schedule, or at least a good one.

The heuristics we use in our models were developed first from ideas which, intuitively, seemed to be reasonable approaches to scheduling, later from additional ideas which grew out of experience in applying the models, and finally from the theoretical considerations discussed in Chapter 3.

The models described in this chapter follow two different approaches to project scheduling. We will refer to the

---

1 See, for example, Clarkson [9], Fischer and Thompson [12], Gere [15], Karg and Thompson [20], Kuehn and Hamburger [25], Tonge [45].

the models as

1) Resource smoothing programs and

2) Limited resource allocating programs.

In the first approach, jobs are scheduled without regard to resource limits. The resulting manpower requirements on peak workload days are then reduced by shifting suitable jobs beyond the peak days. The focus of the program is on the required resources of a given schedule and how these requirements may be leveled. In the second approach, however, the focus is on the available resources, which are serially allocated, day by day, to jobs ordered according to their early start times. The finish date is variable, since it may be extended when jobs are delayed for lack of resources. The rest of this chapter will be devoted to describing in some detail the scheduling models based on these two approaches.

The MS^2 Models

The first of the models programmed were the so-called MS^2 models (for Multi-Ship, Multi-Shop), originally developed for scheduling ship repairs in a Navy shipyard.[1] They are generally applicable, however, to industrial or other types of problems which involve several projects and a number of different skill groups or shops. As the reader will note, the MS^2 models are examples of resource smoothing programs.

Data inputs are minimal: each job in each project is identified by a number j, a time length t, a shop or resource group s, and a set of immediate predecessors. Multi-skill jobs

---

1 The models were the joint development of Mssrs. F. K. Levy, G. L. Thompson, P. R. Winters, and the author. The first of the MS^2 models was reported in [28].

are handled as separate, single-skill jobs; the program is so written as to insure they will all be started on the same day. Each project, consisting of a set of jobs, is identified by a start date and a due date. Below is an outline of two variations of the model describing the heuristics they use for generating a schedule.

MS^2-1

A) Schedule all jobs at early start and plot manpower requirements in each shop, by day.

B) Calculate peak manpower requirements in each shop, and set "trigger levels" for all shops one unit below their respective peaks.

C) Once again start scheduling jobs (in technological order),[1] calculating the manpower loading charts simultaneously. Stop when the trigger level of any shop (call it s) is exceeded.

D) Examine the jobs that are active on the peak day in shop s. Compile a list of jobs which have sufficient slack to move them beyond the peak day without delaying the due date, and arrange them in descending order of their total slack. Pick one of these jobs (by a selection process[2] that favors the jobs highest on the list), and move it to the right on the Schedule

---

1 Jobs are technologically ordered if no job appears in a list until all of its predecessors have been listed.

2 The selection procedure contains random elements and operates as follows: With a probability of $P > 0$, select the first job in the list for the desired operation. If the first job is not selected, place it at the bottom of the list and select the second (now the top) job with the same probability P. Ultimately a job will be selected, as P is greater than zero. The probability of selecting any one job in repeated trials is a function of P and the number of jobs in the list, n. Thus the probability of selecting the i^{th} job is

$$\frac{P(1-P)^{i+1}}{1-(1-P)^{n}} .$$

Chart a random number of days between the minimum move necessary to push the job past the peak day and the maximum move allowed by its total slack.

E) Continue with the scheduling of other jobs and plotting of the manpower loading chart. If additional peaks are generated, apply the procedure of D). If all jobs are successfully scheduled, then lower the trigger levels of all shops one more unit and return to C). If job shifting is not successful in removing peaks below the trigger levels, then restore the previous set of feasible trigger levels and attempt to reduce them shop by shop. As soon as no further reduction in trigger levels is possible, then print out the schedule.

F) Repeat the above process (as many times as is computationally feasible). Because of the random elements in the program, it is likely that different schedules will result from each application of the program. Select as the final schedule the one having the lowest manpower costs (which are assumed to be proportional to the trigger levels--i.e., sufficient men are hired to meet peak loads and are paid whether idle or active on all days).

Note that the program does not assume limited resources. In essence, it attempts to minimize the peak shop requirements given a fixed due date (and is therefore subject to some of the same limitations we noted in Chapter 2 for similar scheduling models). A variation of the above model--MS^2-2 --reverses the constraints; resources are limited and the due date is variable.

MS^2-2

A) Schedule jobs at early start, one at a time, plotting manpower requirements in each shop, by days, until the requirements exceed the established resource limit in some shop.

B) Attempt to right shift slack jobs in the same manner as D) above.

C) If job shifting is successful and a feasible schedule is produced, then print the schedule and repeat the program (as many times as is possible). If manpower requirements cannot be brought below fixed resource limits, then move the due date out one day (or some other increment of time) and return to A). Eventually enough time will be allowed to schedule all jobs without exceeding resource limits.

D) After repeating the program several times, pick the best schedule (in general, the shortest one). Costs here include not only manpower charges but a "penalty fee" for each day the project is delayed beyond the initial due date. (The cost function for penalties need not be linear; an exponentially increasing function might be more appropriate for some project situations.)

Thus MS^2-2 is a limited resource model, but not of type two described above; resources are allocated job by job down a technologically ordered list, rather than day by day to jobs ordered according to their early start times.[1]

---

1 A job list ordered by early start times (in ascending order) is also technologically ordered; but the reverse is not necessarily true.

The SPAR Models

More sophisticated in terms of scheduling heuristics and data handling ability are the SPAR series (Scheduling Program for Allocating Resources) developed by the author. They were specifically designed to consider the constraints of limited resources, the possibilities of variable crew sizes on jobs in a project, and additional alternatives (besides shifting slack jobs) for dealing with peak load periods. The basic program is quite straight forward. The early start (ES) and slack (TS) are calculated for each job in a project, based on technological constraints only. Then jobs are scheduled, day by day, starting with d = 1, by selecting jobs from the list of those currently available (i.e., jobs with ES = d) and ordered according to their slack. The most critical jobs have the highest probability of being scheduled first, and as many jobs are scheduled as available resources permit. (It is possible that not even critical jobs on a given day may be scheduled, because of jobs scheduled on previous days which are still active.) If an available job fails to be scheduled on day d, its ES is increased to d + 1 and an attempt is made to schedule it the next day. Eventually all jobs so postponed become critical and move to the top of the priority list of available jobs. In each day's list of available and active jobs, there are always one or more zero-slack jobs (see Levy, Thompson and Wiest [27]); they receive special treatment in some of the models described below. Probabilistic elements built into the program provide some randomness of job assignments and the likely production of different schedules each time the model is applied to a project--similar in this respect to the MS^2 models.

As is evident from the above description, the GPAR models are examples of the second approach to project scheduling described earlier.

Modifying Heuristics

The basic program described above is enriched by a number of additional scheduling heuristics or subroutines designed to increase the use of available resources and/or decrease the length of the schedule.

A) Crew Size Selection: With each job is associated a normal crew size cs_n (the number of men or other resources normally assigned to the job), a maximum crew size cs_M (the maximum number of men required for "crashing" the job), and a minimum crew size cs_m (the smallest number of men which can be assigned to the job). Normally $cs_m < cs_n$ cs_M. In some cases, jobs can be neither stretched out nor crashed, in which case $cs_m = cs_n = cs_M$; or they can be stretched out but not crashed ($cs_m < cs_n = cs_M$), or vice versa ($cs_m = cs_n < cs_M$). Some jobs may have $cs_m < cs_n < cs_M$ but not permit changes in cs once a given crew size has been assigned; this information is also fed into the model with other job data. The rules for crew size selection are as follows:

1) If the job has zero slack and if resources available a_{sd} are $\geq cs_M$, then schedule j with $cs = cs_M$. If $cs_n \leq a_{sd} < cs_M$, then schedule j with $cs = a_{sd}$. If $a_{sd} < cs_n$, then set $cs = cs_n$ and go to the Borrow and Reschedule routines (described below). If j is still not scheduled, set $cs = cs_m$ and try the same routines.

If j is still not scheduled, then set ES = ES + 1 and attempt to schedule the job the next day.

2) If j has positive slack and $a_{sd} \geq cs_n$, then schedule j with $cs = cs_n$. If $cs_m \leq a_{sd} < cs_n$, set $cs = a_{sd}$ and schedule j. If $a_{sd} < cs_m$, then two approaches are used (according to the particular SPAR model): (a) set $cs = cs_m$ and try the Borrow and Reschedule subroutines, or (b) set ES = ES + 1.

B) Add-On to Critical Jobs: Before any new jobs are scheduled on a given day, examine jobs previously scheduled and still active. If any of these jobs have zero slack and $cs < cs_M$, and if resources are available, increase cs as much as possible up to cs_M. (Jobs which do not permit changes from an initial cs assignment are excluded from consideration in this subroutine.

C) Multi-Resource Jobs: Sometimes a number of different resources (men, machines, etc.) are required for a given job, each of which may be limited in quantity. In such cases, create separate jobs for each resource and assign the jobs to start on the same day, by means of the following device. If n jobs are created out of an n-resource job, then append to each job j in the group the number of the $(j+1)^{th}$ job, except for the n^{th} job, to which append the number of the first job in the group. Thus if n = 3 and the three jobs created are numbered 5, 6, and 7, they would appear in the job list with multi-resource notation as follows:

Job Number	Multi-Resource Reference
-	-
-	-
5	6
6	7
7	5
-	-

(Single resource jobs carry "0" in the multi-resource reference column.) All jobs in a multi-resource group have the same predecessors (and the same successors), so all become available for scheduling on the same day d. As each job j is scheduled, check to see if any other job in the multi-resource group has been postponed to day d + 1. If not, schedule j on day d; if so, then postpone job j by setting $ES_j = d+1$. If an attempt to schedule job j fails because of limited resources, then deschedule all other jobs in the multi-resource group which have already been scheduled, and reset their ES to d+1.

D) Borrow from Current Active Jobs: When $a_{sd} < cs_n$ for some available zero-slack job j (or $< cs_m$ if the job has slack, in some variations of SPAR), the model enters into a subroutine for searching currently active jobs to see if sufficient men might be borrowed from them for scheduling j on day d. In order to qualify for consideration a job k must

1) have $AS_k \leq d$ and $AF_k > d$ (that is, it must be active);[1]
2) use the same resources as j;
3) have $cs > cs_m$;

---

1 After a job has been scheduled, its start and finish times are noted by AS (assign start) and AF (assign finish).

4) be a job for which assigned crew sizes may be altered during the job's active period;

5) have $TS_k^* > TS_j$, where TS_k^* is the slack of k after its crew size has been reduced to cs_m.[1]

Compile a list of such jobs and sort in descending order of TS. Select jobs from this list by the random device described earlier (jobs with the most slack have the highest probability of being selected), and reduce their cs to cs_m until sufficient men are available for scheduling job j. Return then to the main routine and schedule j.

The subroutine also has a "count first and see" feature: check to see that borrowing will actually produce enough men to schedule j before actually making the adjustments. The subroutine is also tied to the Reschedule subroutine (below): if some (but not enough) men can be borrowed, examine the possibility of also rescheduling jobs before abandoning as a failure the borrow subroutine. Finally, if the number of men that can be obtained by borrowing and rescheduling is insufficient to schedule j, then set $ES_j = d+1$.

E) Back Up and Reschedule Active Jobs: Sometimes a job j could be scheduled if other jobs previously scheduled which use the same resources had been postponed to a later start date. In order to qualify for this rescheduling, a job k must

1) have $AS_k \leq d$ and $AF_k > d$. (that is, it must be active);

2) use the same resources as j;

---

1 In the SPAR-1 models (see p. 77), cs is reduced only from day d on; in SPAR-2, the reduction is also retroactive to AS_k.

3) have $TS_k^* > TS_j + b$, where TS_k^* is the slack of k after it has been rescheduled to start on d+1. (b is a parameter which may be used to increase or decrease the number of jobs considered for rescheduling; a large b would result in the consideration only of jobs with large TS^* values as compared to TS_j, reflecting the possibility that such jobs may have to be further postponed [beyond d+1] because of resource limitations.)

The subroutine operates as follows: Sort all jobs which qualify according to the above criteria in descending order of ES, and further sort the list, in each group of jobs with the same ES, in descending order of TS^*. From this list, select jobs to be rescheduled, exhausting jobs whose ES = d-1 before going to jobs with ES = d-2, and so forth.[1] Thus the tendency is to first reschedule jobs which have the most slack and which have to be postponed the fewest number of days. When sufficient jobs have been rescheduled to permit the scheduling of j, then return to the main routine and schedule j.

Note that the rescheduling of a job k does not affect the TS_j of any jobs in the set of available jobs to be scheduled (i.e., jobs whose ES_j = d). TS_j is a function of the early finish of job j's immediate predecessors and of the late start of its immediate successors. However, k belongs to neither set, and it can be delayed up to its LS without affecting jobs in either set. Before k is rescheduled, $AS_k < ES_j < EF_j$, and

---

1 The random selection device is again used.

$AF_k > ES_j$. But $AS_k < EF_j \rightarrow K \notin$ [successors of j], and $AF_k > ES_j \rightarrow k \notin$ [predecessors of j]. The only way rescheduling k could affect TS_j would be to delay the project finish date and hence the late start of all jobs in S_j. According to the routine above, however, k will not be rescheduled unless it has--after rescheduling--at least as much slack as j. Thus no job k will be rescheduled if doing so would extend the finish date. Since the TS of jobs in the available set is not affected by rescheduling, then there is no need to go through the "housekeeping routine" (below) of recalculating new values for ES, LS, TS, etc. after each job is rescheduled; the order in which available jobs are scheduled is thus unchanged.

The reschedule routine has much the same effect as a "look-ahead" feature. Instead of attempting to look ahead to future needs of critical jobs (which would be difficult to do in the limited resource case, since jobs are not always scheduled at their ES), the model schedules all jobs possible as it moves along from day to day, "repenting" of previous scheduling "errors" if jobs are encountered which have more critical need of resources than the jobs to which the resources were assigned at earlier dates.

F) Add-On Unused Resources: After as many as possible of the available jobs are scheduled, there may still be unused resources in some shop s. The program attempts to assign these resources to active jobs. In order to qualify for additional resources, a job j must

1) have $AS_j \leq d$ and $AF_j > d+1$ (it must be active with at least one more day to go);
2) use the resources of shop s;

3) have $cs < cs_M$;

4) be a job for which assigned crew sizes may be altered during the job's active period.

Examine the remaining resources r_s of each shop in turn, after all available jobs have been considered. If $r_s > b_s$ (a parameter reflecting the per cent utilization of resources which is desired), then compile a list of jobs which meet the above criteria. Order the list in ascending order of TS, pick a job (by the random device, favoring the jobs with the lowest TS), and increase its cs to cs_M (or to $cs + r_s$ if $cs_M - cs > r_s$). Continue increasing the cs of jobs in the list until the list or r_s is exhausted. The increment is temporary; jobs so supplemented return to their assigned cs the next day (unless unused resources are available then also).

The program contains three additional major subroutines which do not properly belong in the above list of modifying heuristics. They are listed below, however, for completeness in describing the program.

G) Housekeeping Subroutine: After going through the above scheduling routines each day d, the model then records the results in a manpower loading table (resources used in each shop) and in a job assignment table (number of men assigned on each job); and it updates the critical path data (ES, LS, EF, LF, AS, AF, TS, and man days remaining) for each job. Note that several of the above subroutines may alter a job's total slack; e.g., when cs_M is assigned to a critical job, the shortening of the job may result in its gaining positive slack (and some other job or jobs becoming critical).

H) Cost Calculations: The Functional: In all of the SPAR models, the finish date is a variable; and in those with a search routine (described below), shop resource levels are also variable. The functional can take any desired form, but two have been used thus far:

1) $$\text{Total Cost} = K \cdot z + \sum_{s=1}^{m} q_s^* \; w_s,$$

where K is a daily cost (e.g., overhead expenses and/or due date penalties, charged on a per-day basis), z is the length of the schedule, q_s^* is the maximum crew size required in shop s, and w_s is the average daily wage in shop s. The implicit assumption is that crew sizes are maintained at peak loads and are paid whether active or idle. Should the circumstances of a specific project justify it, a non-linear cost function of z could be substituted for the linear one above--e.g., when penalties for exceeding the due date increase exponentially.

2) $$\text{Total Cost} = K \cdot z + \sum_{s=1}^{m} a_s^* \; w_s,$$

where K, z and w_s are as above, and a_s^* is the optimum shop crew size based on a premium rate v for overtime or subcontracting when manpower requirements exceed a_s^*. (v is a multiplying factor; thus overtime would cost $v \cdot w_s$ per day.) The assumption is made that regular crew sizes are maintained at a_s^*, even during slack periods. If we let F_{a_s} represent the number of days out of z that manpower requirements in shop s equal or exceed a_s, then a_s^* is that manpower level for which

$$F_{a_s^*} \cdot v \geq z > F_{(a_s^*+1)} \cdot v$$

That is to say, if a_s is raised one day above a_s^*, then the regular wage rate paid the extra man for z days ($z \cdot w_s$) exceeds the costs of paying a man at overtime rates for $F_{(a_s^*+1)}$ days ($F_{(a_s^*+1)} \cdot v \cdot w_s$).

I) Shop Resource Level Search Rules: It is possible that, in some cases, the costs of increasing shop resource levels would be more than offset by the resulting decrease in overhead charges and/or due date penalty fees. The reverse situation might also be true. Hence we have developed some search rules for trying to find some optimum combination of shop resource levels and resulting finish date. Two approaches have been explored:

1) Start with minimal resource levels (just sufficient to insure that all jobs can be scheduled, serially if not in parallel). Generate a schedule and calculate its cost. Then increase the resource level in all shops associated with jobs having zero slack. Generate a new schedule and calculate its cost. If lower than the first schedule's cost, repeat the same process. If higher, restore the previous resource levels and increase them shop by shop, selecting shops associated with jobs having zero slack (or, alternately, shops with the most "filled-up" days). Keep raising resource levels in a given shop until schedule costs increase; then restore the previous best level and try a new shop. When all shops have been considered, then select the best schedule of those previously generated and stop.

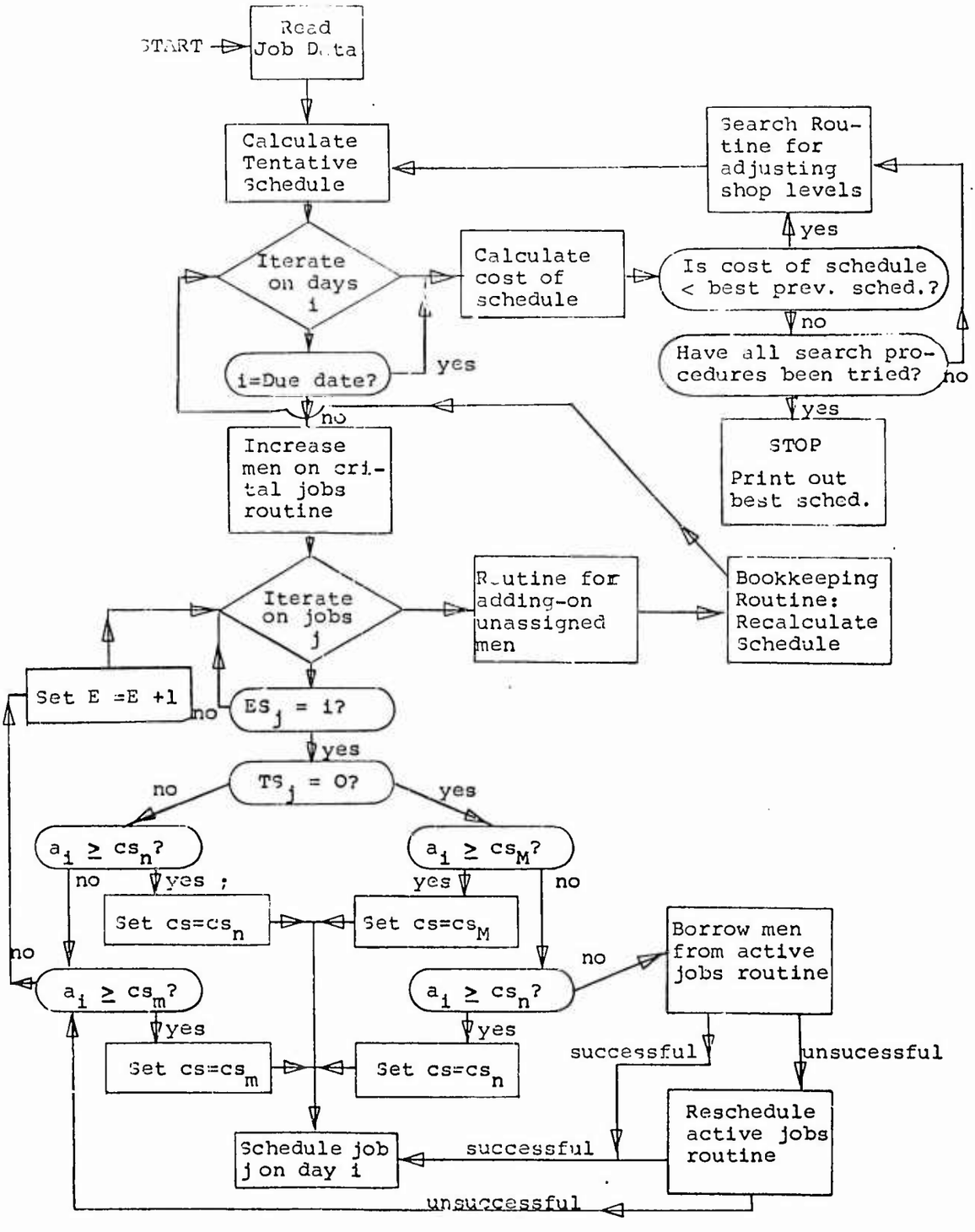
2) Start with ample resource levels (e.g., levels high enough to just permit starting all jobs at ES). Generate a schedule and calculate its cost. Then decrease the resource levels of all shops one unit and generate a new schedule. If its cost is lower than that of the first schedule, reduce resource levels again. Repeat until the cost increases, then restore the previous best levels. Next reduce shop levels one shop at a time, picking first the shop with the fewest "filled-up" days. Continue reducing the level in each shop as long as an improvement in schedule cost results. After reductions have been attempted in all shops, then pick the best previous schedule and stop.

SPAR-1

The first of the SPAR models is outlined below in a simplified flow diagram (Figure 4). It uses the above scheduling heuristics and may be modified to handle single or multiple projects, fixed or variable crew sizes, constant or variable shop limits (over the scheduling period), and various functionals and search routines as described above. These modifications (all of which have been programmed and tested) are useful in fitting the model to a particular project situation.

Figure 4

FLOW DIAGRAM - SPAR-1



SPAR-2

SPAR-2 is a major modification of SPAR-1 and is based on the concepts discussed in Chapter 3. The first part of SPAR-2 is quite similar to SPAR-1, with the following exceptions and restrictions: (a) all jobs are initially started at cs_n; (b) resource levels are assumed to be constant over the schedule period (to assure the existence of a critical sequence); (c) there is no add-on routine, and the borrow routine adjusts the cs of job k from ES_k rather than from day d only (to assure that constraints to left and right shifting occur only at the terminal points of jobs); (d) a multi-resource job, while divided into separate jobs for programming purposes, is still considered a single job when determining if a schedule is left-justified. Note that a job in a multi-resource group may be neither resource nor technologically constrained (and hence, by itself, not left-justified), but restrained only because it must start at the same time as another job in the group which is resource constrained.

SPAR-2 first generates a schedule that is left-justified (essentially by the SPAR-1 routine). Then an associated right-justified schedule is generated (according to the rules described in Chapter 3) and conditional slack values calculated. A recursive search is made for local suboptimalities which, if found, are removed. In the resulting left-active schedule, the critical sequence(s) of jobs is identified, and an attempt is made to shorten the schedule by crashing one or more of these jobs. Two criteria are used in selecting jobs to crash: first, jobs are picked whose cs can be increased to cs_M without exceeding the shop limits (or with the least required overtime), and, in

case of ties, the job is selected whose cost-time curve has the least slope (after the manner of Kelley [22]). After a critical job is crashed, a new schedule is generated and the above routine applied again--repeating the process as long as there are improvements in the cost of the schedule.

A simplified flow diagram of SPAR-2 follows in Figure 5.[1]

A Comparison of RAMPS and SPAR

Having described the basic SPAR program, we are now in a better position to compare it with CEIR's scheduling model, RAMPS. The two programs represent independent, parallel efforts to solve the same problem; each was completed before its author(s) had any knowledge of the other. Both take a similar approach to scheduling, but there are basic, important differences in the programs, and each can claim some advantages not enjoyed by the other.

First we will note the similarities in data-handling ability and design. Both SPAR and RAMPS

1) employ a network technique for determining technological ordering of jobs;

2) use deterministic rather than probabilistic time estimates;

3) consider three resource utilization rates for each job, with associated job times;

4) consider resource availabilities, which may vary during the scheduling period;

5) handle multiple projects, each with a different due date;

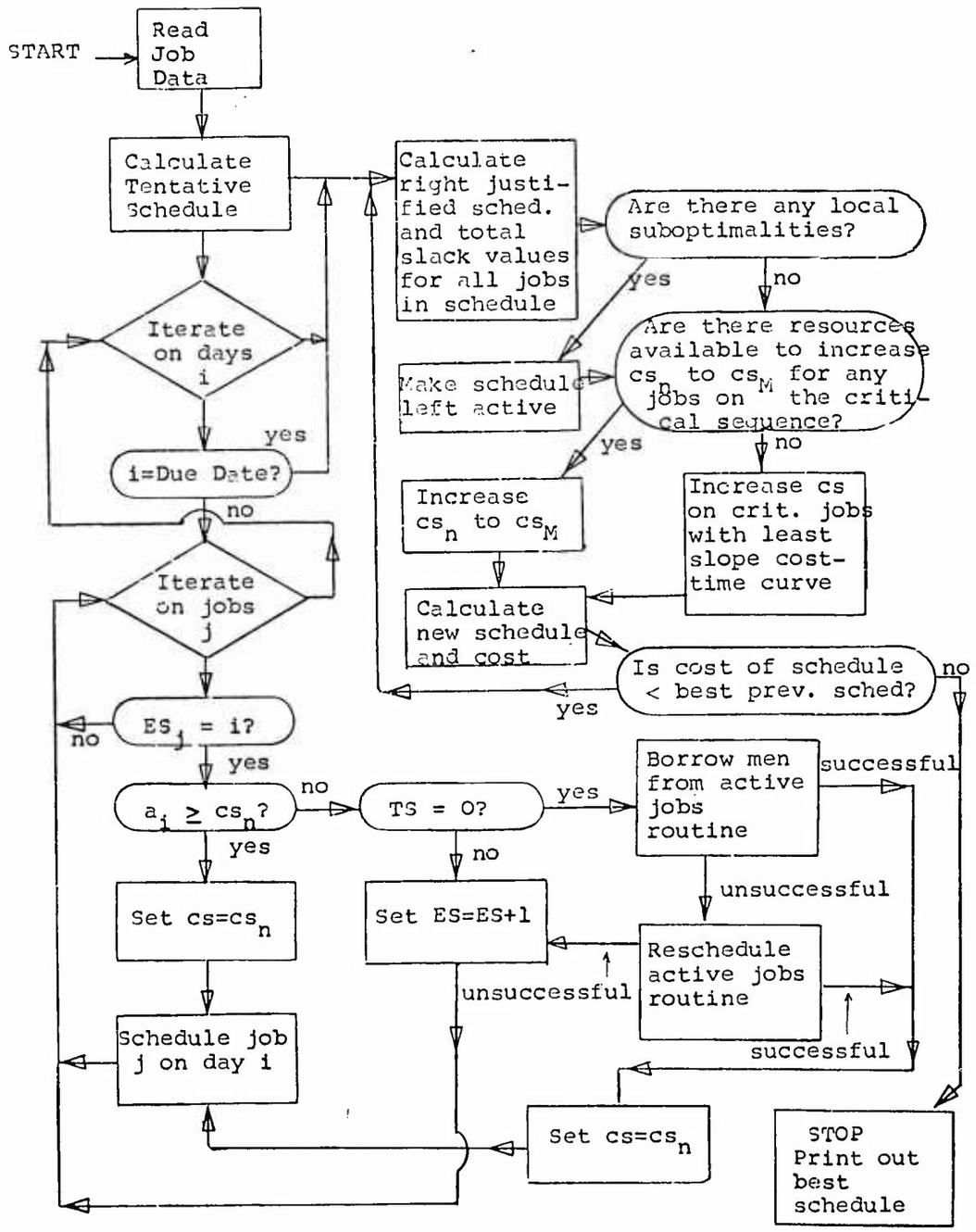
6) allow resource teaming on jobs;

---

1 At the time of this writing, SPAR-2 was being programmed for the Bendix G-20. "Debugging" and trial runs had not been completed.

Figure 5

FLOW DIAGRAM - SPAR-2



7) consider costs of normal time, idle time, overtime (or other premium resources), and due date penalties;

8) have approximately the same job-handling capacity.

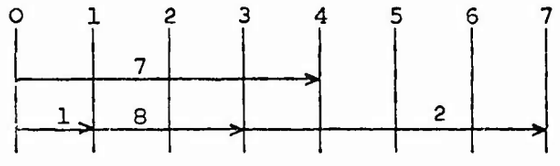
Furthermore, both are "type 2" models (see p. 63); i.e., they serially allocate available resources, day by day, to jobs which are available for scheduling. Due date is variable and a function of given limited resources. Beyond these similarities, there are important differences in the programs, a few of which we note.

On each day examined, RAMPS schedules resource by resource, starting with the one in most critical demand. (Each resource has a "criticality index," calculated before any scheduling is done, which is based on the total man-days required for a resource [summed over all jobs and all days] and the man-days of that resource available. The index is unchanged by the scheduling process.) The program generates, for each resource on each day, all non-trivial assignment patterns that exist (i.e., all feasible combinations of current jobs), given the jobs available for scheduling on that day (including jobs already started and still active), the three utilization rates for each job, and the resources available. Each assignment pattern is scored according to functional equations which include as variables total float, free float, number of jobs scheduled, number of jobs split, idle resources, and criticality (in terms of resource needs) of immediate successor jobs. The pattern with the highest score is selected, and the program proceeds to the next resource, or to the next day. Thus costs are minimized each day (but not necessarily over the whole schedule period). Once a job is scheduled, it remains so. There are no provisions for adjusting earlier job assignments. The program is completely deterministic; it

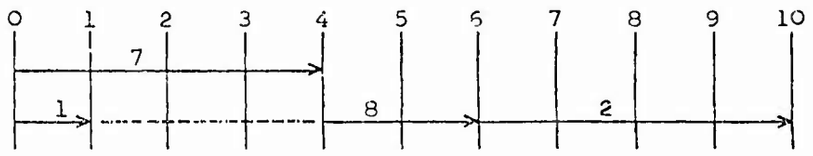
contains no probabilistic elements in its job-selection routine.

We observe, then, that RAMPS has a more elaborate decision-rule for selecting jobs to be scheduled than has SPAR. The greater number of variables considered gives it some flexibility, for purposes of studying the effects of these variables, not enjoyed by SPAR. Additionally, RAMPS considers not only three utilization rates for each job, but also different work efficiencies associated with these rates. SPAR-1 implicitly assumes that "crash" and "slow-down" rates are as efficient as the normal rate. (SPAR-2 takes into account possible inefficiencies of non-normal rates in its cost function.) RAMPS thus reflects the U-shaped cost curve commonly described by economists in their analysis of production functions.

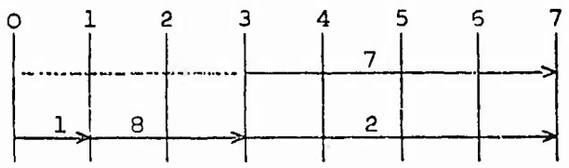
On the other hand, SPAR has some important features not paralleled in RAMPS--e.g., the add-on, borrow, reschedule and search routines. While a RAMPS schedule is "optimal" each day, one can easily think of projects for which day-by-day optimizing will lead to a suboptimal overall result. Consider for example the following schedule graph of a project in which all jobs require the same resource (the numbers indicate the amounts of resource required by each job):



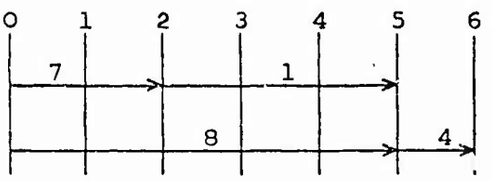
If the resource limit is 10, and if job-splits are disallowed (by a high penalty for such splits), RAMPS would generate the following schedule:



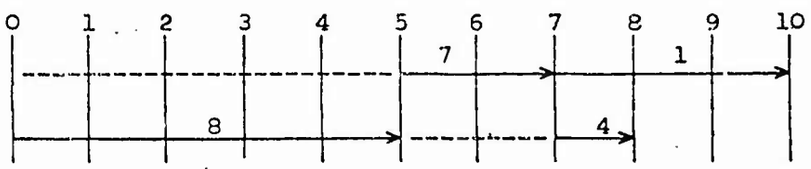
with a due date of 10. While SPAR would originally schedule the 7 and the 1 jobs on day 0, the Reschedule Routine would lead it to postpone job 7 and generate a seven-day schedule, as follows:



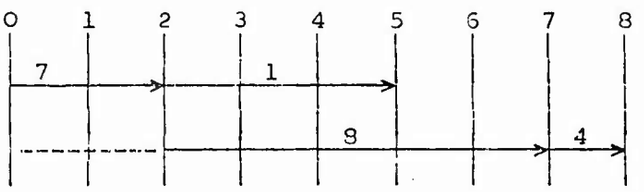
The probabilistic elements or SPAR models also aid them in finding optimal schedules which would otherwise be missed. Assume, for example, the following project:



With a resource limit of 10, both RAMPS and SPAR (if it had no probabilistic elements) would generate the following ten-day schedule (job splits are again disallowed):



Because of the random elements in the job-selection routine, however, in repeated applications SPAR would sometimes schedule job 7 first and find the following optimal schedule:



Thus the random elements generally increase the probability of finding an optimum schedule (though they do not guarantee that such a solution will in fact be found [the probability may be very small] or even that the probability of finding an optimal schedule is always positive).

Not only does SPAR seek an optimum schedule for a given set of shop limits, but by means of its Search routine, it also seeks more efficient combinations of shop limits and resulting due dates.

Finally, SPAR-2 incorporates the critical sequence concepts developed in Chapter 3. We anticipate that a model which explicitly takes account of the effects of limited resources on critical path analysis should prove superior to a model--similar in other respects--which does not. We have yet to test this hypothesis, however.

We observe, in summary, that RAMPS and SPAR are similar in data-handling ability and in their general approach to scheduling; but they each have unique features which are interesting and deserving of experimentation. We expect that both will advance our ability to deal more adequately with the large project problem, and that experience gained from their applications will lead to more powerful heuristics for project scheduling.

Some Comments on the Certainty Assumption

We might well be asked if the practical usefulness of our models (a point we have often stressed) is not significantly lessened by our certainty assumption. Would not our schedules be more realistic and useful if we recognized the probabilistic nature of job times? Our answer is threefold:

1) Job times themselves are often difficult to estimate, let alone their probability distributions. A frequent criticism made of PERT, which uses a simple, three-point distribution of job times, is that the "pessimistic time" estimates are often too pessimistic, biasing downward the calculated values for "expected time." The value of probabilistic times and computed figures for variance is questionable, it seems to us, when little is known about the probability distributions for job times.

2) Even if reasonably accurate probability distributions were available, the added requirements on computer space and time to incorporate such data in the models would have necessitated the omission of other project data (such as variable job times), or restrictions in the size of projects that could be scheduled, or a simpler program in terms of scheduling heuristics. We felt the potential advantages of incorporating probabilistic times were not worth the above sacrifices in the scheduling models.[1]

---

1 The advantage of using probabilistic times in the related problem of job shop scheduling was questioned in a study by Muth [35]. He concluded that "the schedule span is not very sensitive to moderately large errors in estimated job times."

3) The main use of probabilistic job times in PERT-type models is for calculating an expexted project completion time, with an associated variance. If this information is of particular value to the user of a project schedule, then it would be relatively simple to generate such figures after the models described earlier have developed a schedule (based on deterministic job times). In PERT fashion [30], expected times for jobs along the critical path (or critical sequence) could be calculated from the probability distributions of these jobs and added to obtain the expected due date for the project, with variance being determined similarly. Or a simulation approach could be used (as reported by Levy and Wiest [29][1]) in which job times are selected from relevant probability distributions by Monte Carlo techniques and project due dates calculated. From the resulting distribution of due dates, the expected due date and variance can be determined, for whatever managerial purposes they seem useful.

---

1 The simulation approach described in [29] assumes that all jobs are started at early start. The method could still be used in projects with limited resources if jobs along the critical sequence only are considered. As in the PERT approach, this ignores the possible effects of near-critical jobs on the expected due date and variance.

Chapter 5

OPERATING RESULTS FROM THE PROJECT SCHEDULING MODELS

Computer Requirements of the Scheduling Models

All of the models discussed in Chapter 4 have been programmed in 20-GATE, an algebraic language for the Bendix G-20 Computer at Carnegie Institute of Technology. In its latest formulation, the MS[2] program requires approximately 3,000 machine locations, leaving about 22,300 locations for data.[1] Thus it can handle a project with up to 1,030 multi-resource jobs in 12 shops, extending over a scheduling period of up to 300 time periods. Alternately the program can handle 2,650 single skill jobs in 12 shops over the same scheduling period. Trade-off between the number of jobs, shops, and time units may be made according to the following formula for the number of machine locations (W) required for data:

W = (No. of multi-resource jobs)(6 + No. of shops) + (No. of days)(No. of shops) + 75.

The most recent SPAR-1 program, including necessary computer subroutines, requires approximately 6,000 machine locations, leaving about 19,200 locations available for data in core storage. Data space requirements of a project can be computed as follows:

W = (No. of single-resource jobs)(14) + (No. of nodes)(2) + (No. of shops)(No. of time periods) + 1,200.

For example, the model could be dimensioned to handle a project with 940 single-resource jobs, 500 nodes and 12 shops

---

1 The Bendix G-20 at C.I.T. has core storage of 32 K single precision words. The algebraic compiler and subroutines occupy the space not accounted for above.

over a time span of 300 days. Thus the job handling ability of MS^2 is somewhat greater than SPAR-1, due to the longer program of the latter and greater amount of job data considered by it.

It would be possible, of course, to program both models more efficiently by using machine language; and their data-handling ability could be increased considerably by more extensive use of tapes, "packing" of data, and other programming devices. Larger and faster computers than the Bendix G-20 would further extend the range of projects which could feasibly be scheduled by the models.

Project Scheduling Experience

Operating experience with the models has been derived largely from their application to four different projects, which we may identify as follows:

Project A: A fictitious project, formulated in part from a random number table, containing 55 jobs performed ibn 4 shops.

Project B: A fictitious project similar to A, containing 100 jobs performed in 4 shops.

Project C: A construction project, containing 97 multi-resource jobs (or 181 single-resource jobs) involving 12 different resource groups ("shops").

Project D: An accounting project, consisting of 691 jobs required for the month-end closing of accounting records in a large manufacturing company; the jobs involve 20 different skill groups ("shops").

We will consider the results of applying the models to these projects, each in turn.

Projects A and B

The initial results of applying MS^2-1 to Project A are reported elsewhere (Levy, Thompson and Wiest [28]); these results are summarized in Figure 6 below. Thus with a project due date of 47,[1] shop limits were reduced from their original levels (with all jobs plotted at early start) as follows:

	Maximum Resource Requirements			
	Shop 1	Shop 2	Shop 3	Shop 4
All jobs at early start	24	18	16	24
After smoothing with MS^2-1	9	8	9	9

The due date of 47 assured that the jobs would have generous amounts of slack, in order to give the program some flexibility in moving jobs. (In real-life projects, such flexibility would generally accrue from the large number of jobs involved, rather than from ample due dates.) Running time for the program was approximately three minutes.

The minimum shop limits reached by MS^2-1 were used as fixed resource limits in the SPAR-1 model which was then applied to Project A. To produce results comparable to MS^2-1, job manpower assignments were first assumed to be non-variable (i.e., "crashing" or "stretching" jobs was not permitted). The resulting schedule is shown below in Figure 7 along with the MS^2-1 schedule. Thus SPAR generated a schedule only 39 days in length, a considerable improvement in this instance over the MS^2-1 schedule. Running time for the program was just under 1 minute.

When variable crew sizes were allowed, a further improvement in the schedule was obtained. As indicated by Figure 8,

1 Throughout this chapter, projects are assumed to start on day 0, so that the schedule length is the same as the due date.

Figure 6

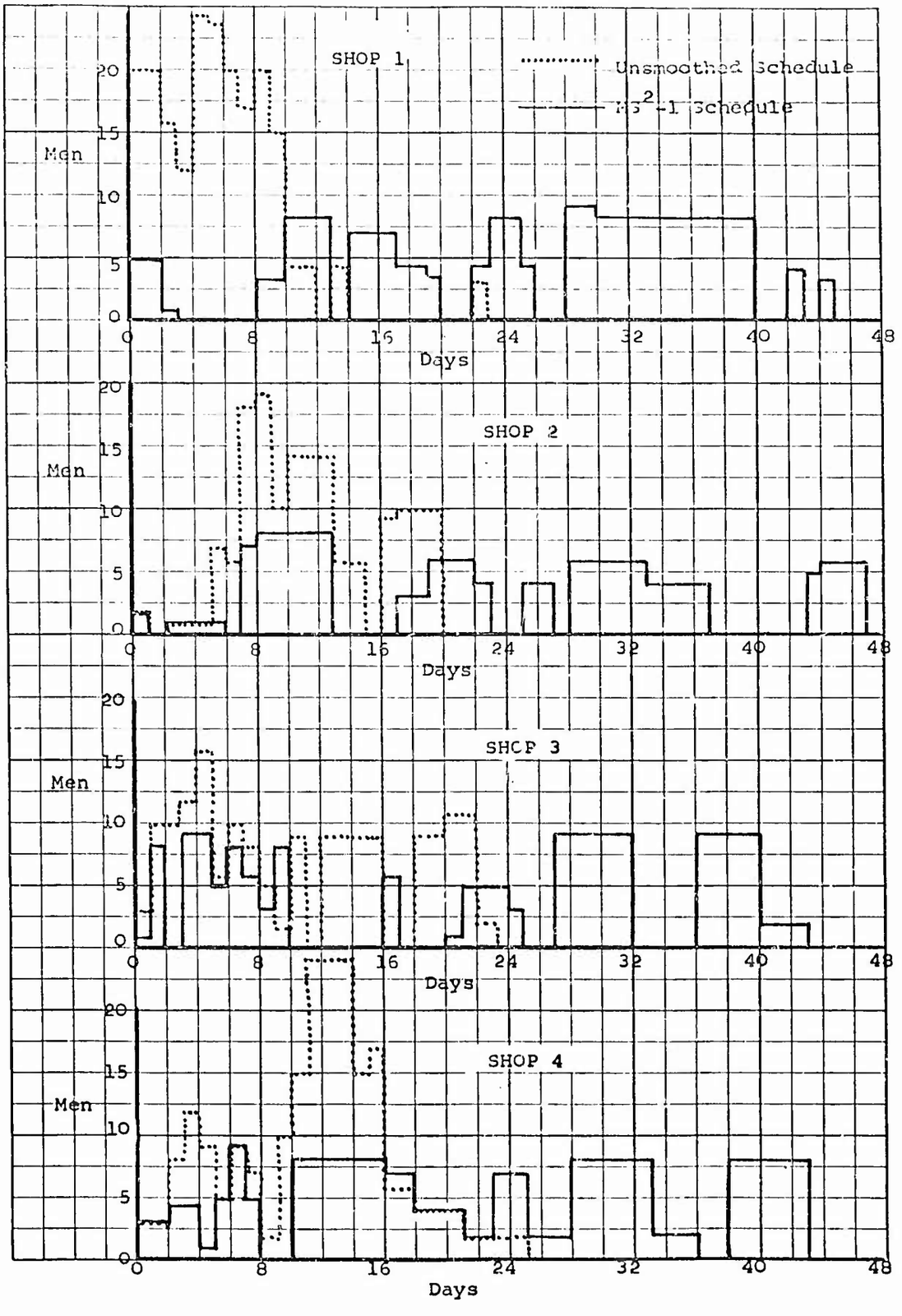

SHOP 1
Unsmoothed Schedule
Ms^2-1 Schedule
Men
Days
SHOP 2
SHOP 3
SHOP 4
0
5
10
15
20
8
16
24
32
40
48

Figure 8

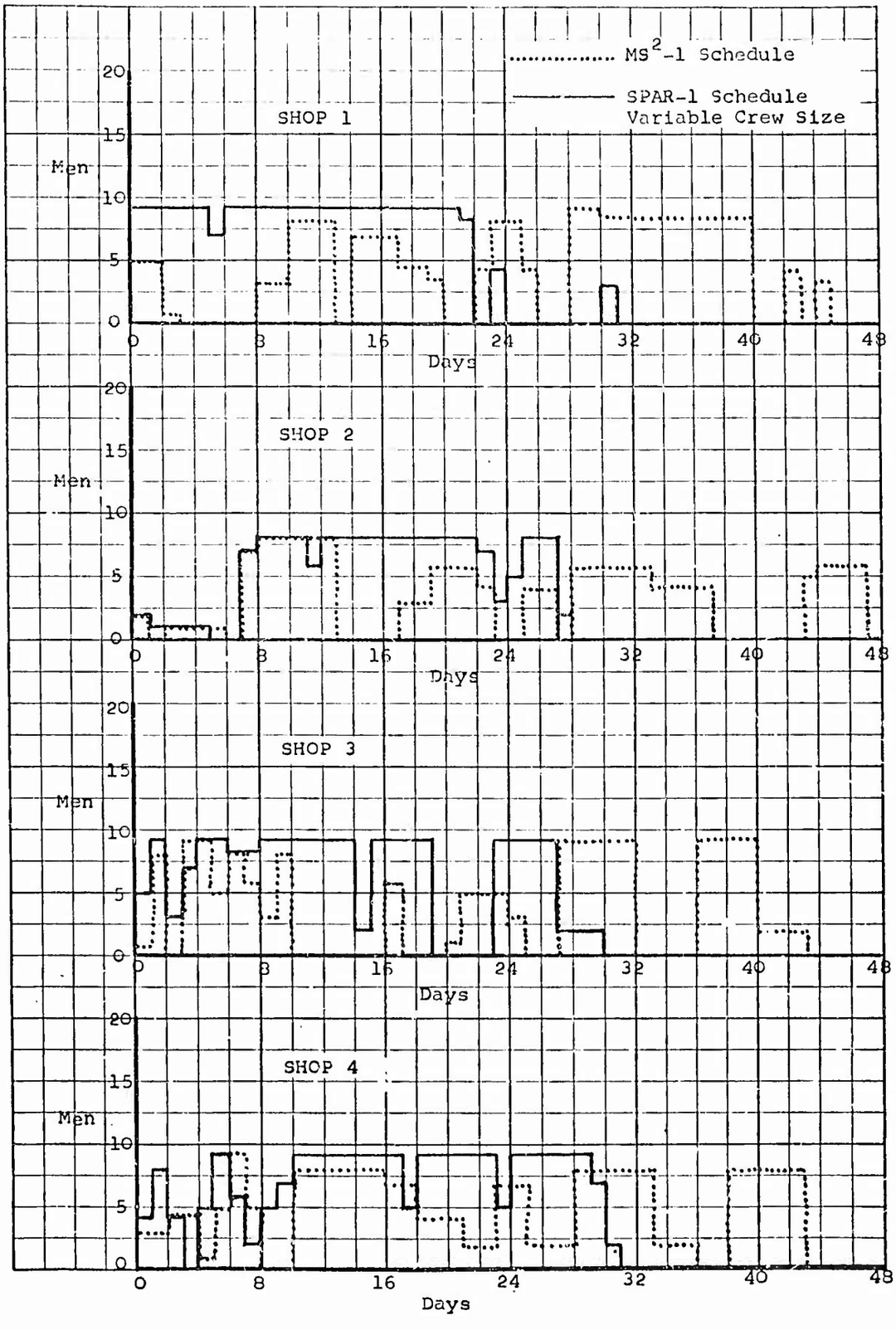
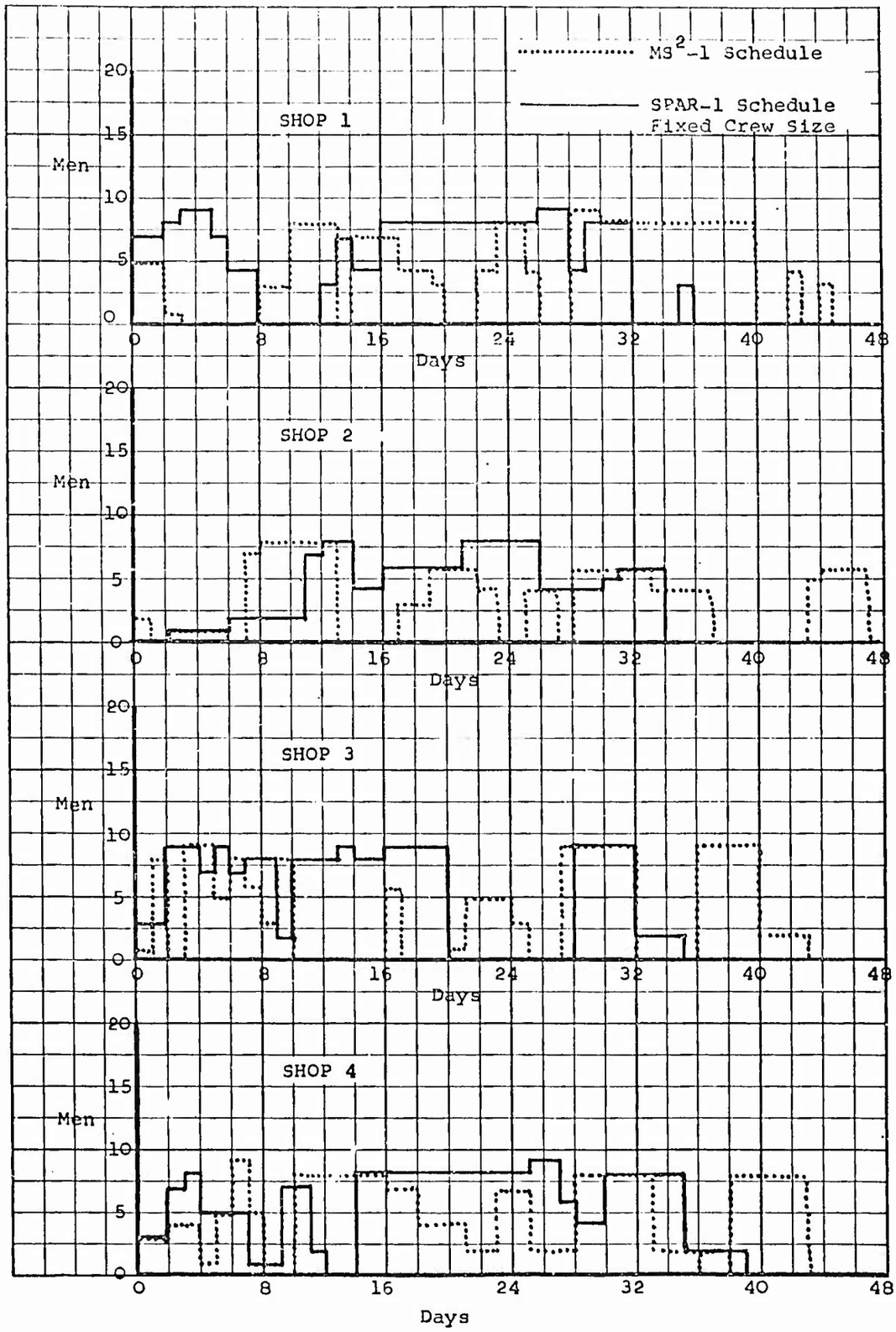

MS^2-1 Schedule
SPAR-1 Schedule
Variable Crew Size
SHOP 1
SHOP 2
SHOP 3
SHOP 4
Men
Days
0
5
10
15
20
0
8
16
24
32
40
48

Figure 7


MS²-1 Schedule
SPAR-1 Schedule
Fixed Crew Size
SHOP 1
SHOP 2
SHOP 3
SHOP 4
Men
Days
0
5
10
15
20
8
16
24
32
40
48

project length was reduced to 31 days.[1]

As noted earlier, the MS^2-1 model attempts to minimize shop limits given a fixed due date; the SPAR models can seek for an optimum combination of shop limits and due date (with delay penalties). Two SPAR-1 programs, each with a different search routine (see p. 76), were applied to Project A with the following results:[2]

Table 1

SPAR-1 Applied to Project A: 55 Jobs
(Search Routine 1: Increasing Shop Limits)

	Shop Limits					
Iteration	Shop 1	Shop 2	Shop 3	Shop 4	Due Date	Total Cost
1	9	9	9	9	33	41,780
2	9	9	9	10	30	39,023
3	9	9	10	11	30	40,832
4	9	9	10	10	32	42,408
5	9	9	9	11	31	41,568

1 The schedule reported here is the best of seven iterations (each requiring about 50 seconds including print-outs); results ranged from 31 to 35 days.

2 Labor costs were assumed to be as follows:

Shop Number	Dollars/Hour
1	3.50
2	4.75
3	2.80
4	5.00

In addition, an overhead charge (including delay penalty) of $150/day was assumed.

Table 2

SPAR-1 Applied to Project A: 55 Jobs

(Search Routine 2: Decreasing Shop Limits)

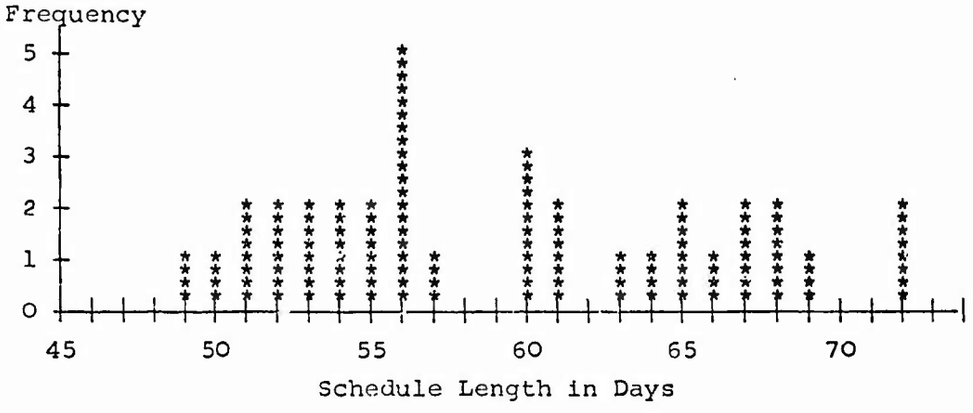
Iteration	Shop Limits				Due Date	Total Cost
	Shop 1	Shop 2	Shop 3	Shop 4		
1	15	15	15	15	24	47,748
2	14	14	14	14	24	44,795
3	13	13	13	13	26	45,480
4	13	14	14	14	24	44,151
5	12	14	14	14	26	47,290
6	13	13	14	14	24	43,277
7	13	12	14	14	24	42,403
8	13	11	14	14	24	41,529
9	13	10	14	14	27	45,958
10	13	11	13	14	24	41,014
11	13	11	12	14	25	42,260
12	13	11	13	13	26	43,580

Due to the nature of the search routines, the search for optimum shop levels stops when a local optimum has been found. Note that the lowest cost schedules found by the two routines differ by only 5 per cent in cost, but that the due dates and shop levels are quite different. Quite probably there are other combinations of shop limits and due dates in the same vicinity that would result in similar (perhaps lower) total costs; repeated applications of the program would likely discover some of these combinations, due to the probabilistic elements in the program.

Results from the application of MS^2-2 to Project A were generally less favorable than the SPAR results. Repeated

applications of MS^2-2 to Project A with resource limits of nine men in each shop resulted in schedules whose lengths are plotted in the following frequency distribution:

Figure 9



Thus the shortest schedule produced was 49 days long, found once in 35 iterations of the program (each iteration required about 1.7 minutes). By contrast, a single application of SPAR-1 to Project A with the same shop limits produced a 34 day schedule in less than a minute of computer time. A second application of MS^2-2 to Project A with resource limits of 10 in each shop resulted in a minimum schedule length of 44 days (nine iterations); SPAR-1, with the same shop limits, generated a 29 day schedule.

Although experience with MS^2-2 was limited to applications to Project A, the results were so clearly inferior to SPAR-1 results (both in terms of schedule lengths and computational time required) that the MS^2-2 approach to resource allocation was abandoned in favor of the SPAR-1 models.

Project B was devised to increase operating experience with SPAR-1 on a larger project and to test the multi-project feature incorporated in that model. The major results can be

summarized briefly in the following tables:

Table 3

SPAR-1 Applied to Project B: 100 Jobs, One Project[1]

(Search Routine 1: Increasing Shop Limits)

Iteration	Shop Limits				Due Date	Total Cost
	Shop 1	Shop 2	Shop 3	Shop 4		
1	10	10	10	10	51	71,700
2	10	11	10	10	47	67,712
3	10	12	10	10	46	67,950

(Search Routine 2: Decreasing Shop Limits)

Iteration	Shop Limits				Due Date	Total Cost
	Shop 1	Shop 2	Shop 3	Shop 4		
1	13	15	14	13	41	71,402
2	12	14	13	12	41	66,651
3	11	13	12	11	42	68,093
4	11	14	13	12	41	64,004
5	10	14	13	12	44	72,945
6	11	13	13	12	42	69,078
7	11	14	12	12	44	74,769
8	11	14	13	11	41	64,135

In this application, Search Routine 2 produced the best schedule in terms of total costs (iteration 4). Running time per iteration was about 1.5 minutes.

For the next run, Project B was divided into a 45 job project and a 55 job project.

---

1. Shop rates and overhead charges are the same as for Project A.

Table 4

SPAR-1 Applied to Project B: 100 Jobs, Two Projects

(Search Routine 2: Decreasing Shop Limits)

	Shop Limits				Due Date		Total
Iteration	Shop 1	Shop 2	Shop 3	Shop 4	Proj. 1	Proj. 2	Cost
1	13	15	14	13	32	44	87,107
2	12	14	13	12	32	46	85,164
3	11	13	12	11	35	49	84,819
4	10	12	11	10	42	53	85,835
5	10	13	12	11	45	45	78,444
6	9	13	12	11	44	52	88,295
7	10	12	12	11	46	49	83,301
8	10	13	11	11	47	45	80,679

As can be noted, the shop levels of iteration 5 produced the lowest cost schedule.

Evaluation of Results: Projects A and B

The most important result to come from the above applications (and others not recorded here) was the demonstration of the models' feasibility in project scheduling. Protocols of the problem-solving process indicated that the various subroutines were operating as planned, and the rather short computational times required gave encouragement for application of the models to larger projects.

Whether the schedules produced were "optimal" by some criterion function is more difficult to access. Hand simulation of the project with variable crew sizes and variable shop limits is, for all practical purposes, impossible; the number of combinations to investigate is too large. However, when crew sizes

and shop limits are held constant, the measure of a schedule's optimality is simply the length of the schedule; and for Project A, at least, hand simulation is feasible. The author performed two such simulations with different shop levels, and compared them with corresponding SPAR-1 schedules. Hand simulation of Project A with shop limits of 9, 8, 9, and 9, in shops one through four, respectively, indicated that the optimum schedule has a due date of 37 days; this compares to a due date of 39 in the schedule produced by SPAR-1. The SPAR-1 schedule for Project A when shop limits were 10, 9, 10, and 10, however, was optimal at a due date of 34. While it would be foolhardy to draw any strong conclusions from these two examples, the results were encouraging, at least.

Project C

Both MS^2-1 and SPAR-1 were applied to Project C, an actual construction project for which job data had been collected and a network drawn by the contractor. The length of the critical path through the network, with all jobs at normal crew size, was 94 days. With this as a due date, the project was first scheduled by the MS^2-1 model in an attempt to level resource requirements.[1] The results are shown in Figure 10, which displays the manpower loading charts for the 12 resources, comparing manpower requirements with all jobs scheduled at early start with requirements after smoothing by MS^2-1.

---

1 The author is indebted to Jack Trawick for the results of this application of the MS^2-1 model. For Project C, Mr. Trawick reprogrammed the model to use computer time and memory space much more efficiently than the earlier formulation.

Figure 10

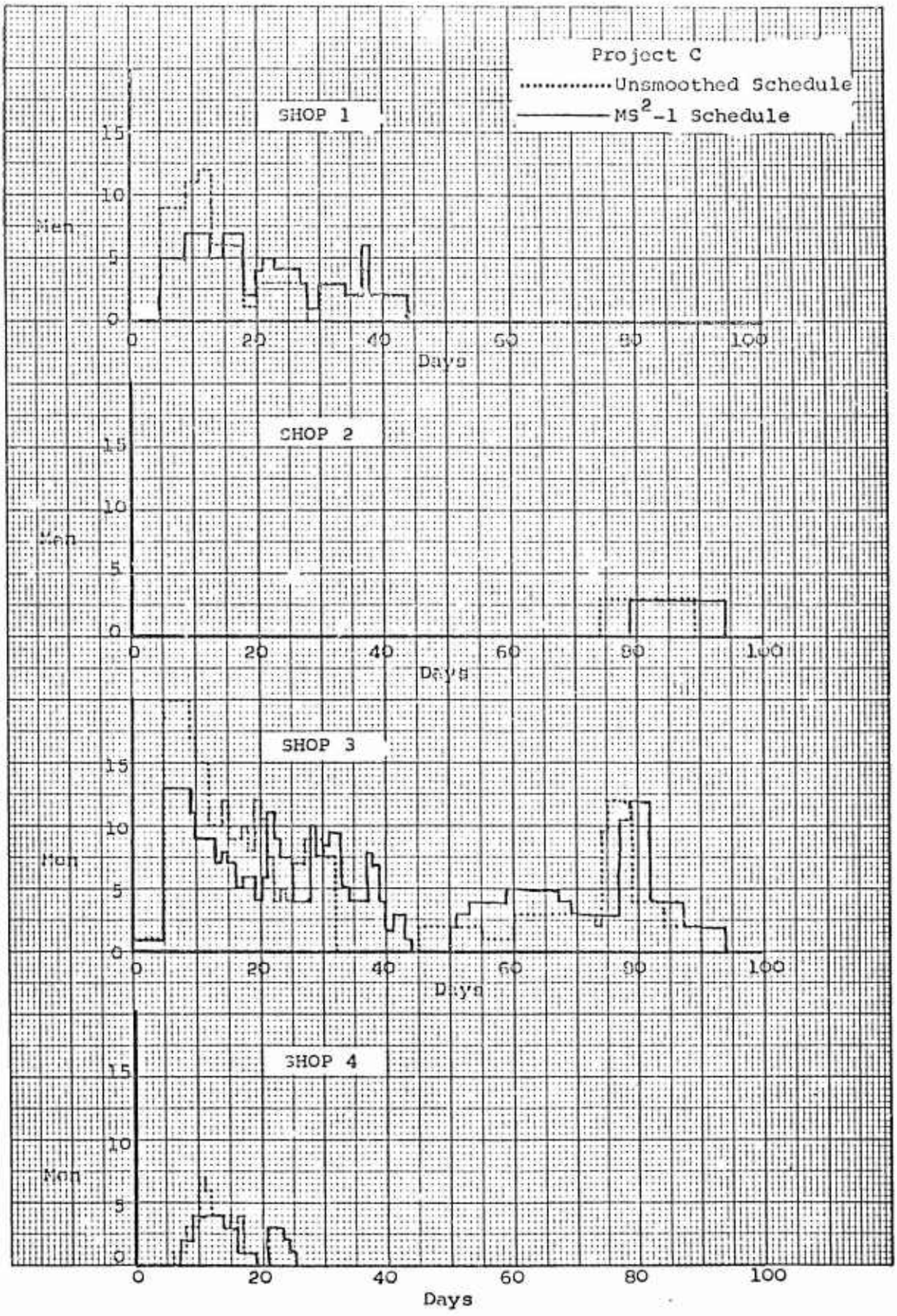

Project C
Unsmoothed Schedule
MS²-1 Schedule
SHOP 1
SHOP 2
SHOP 3
SHOP 4
Men
Days
0
5
10
15
20
40
60
80
100

Figure 10 (cont.)

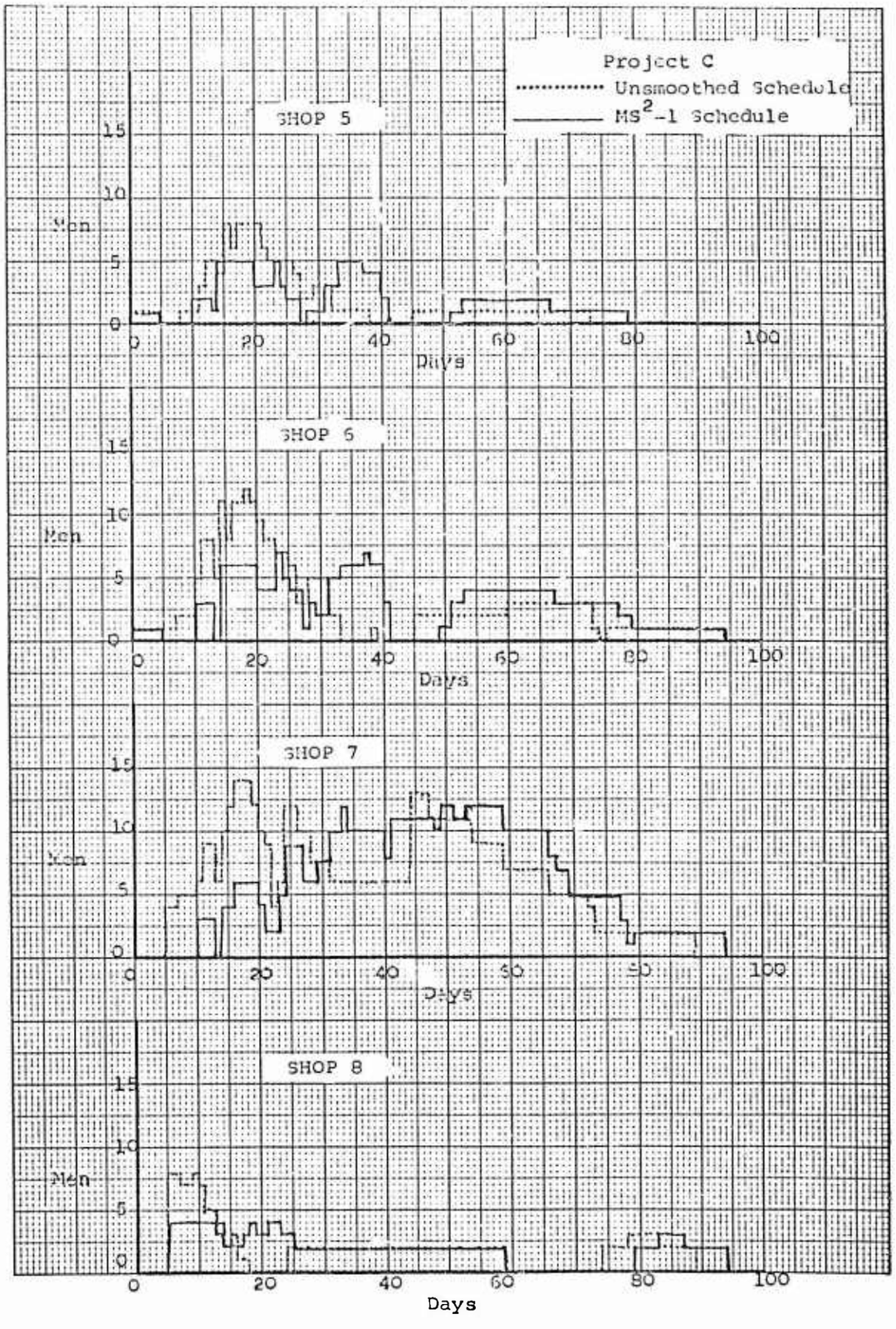
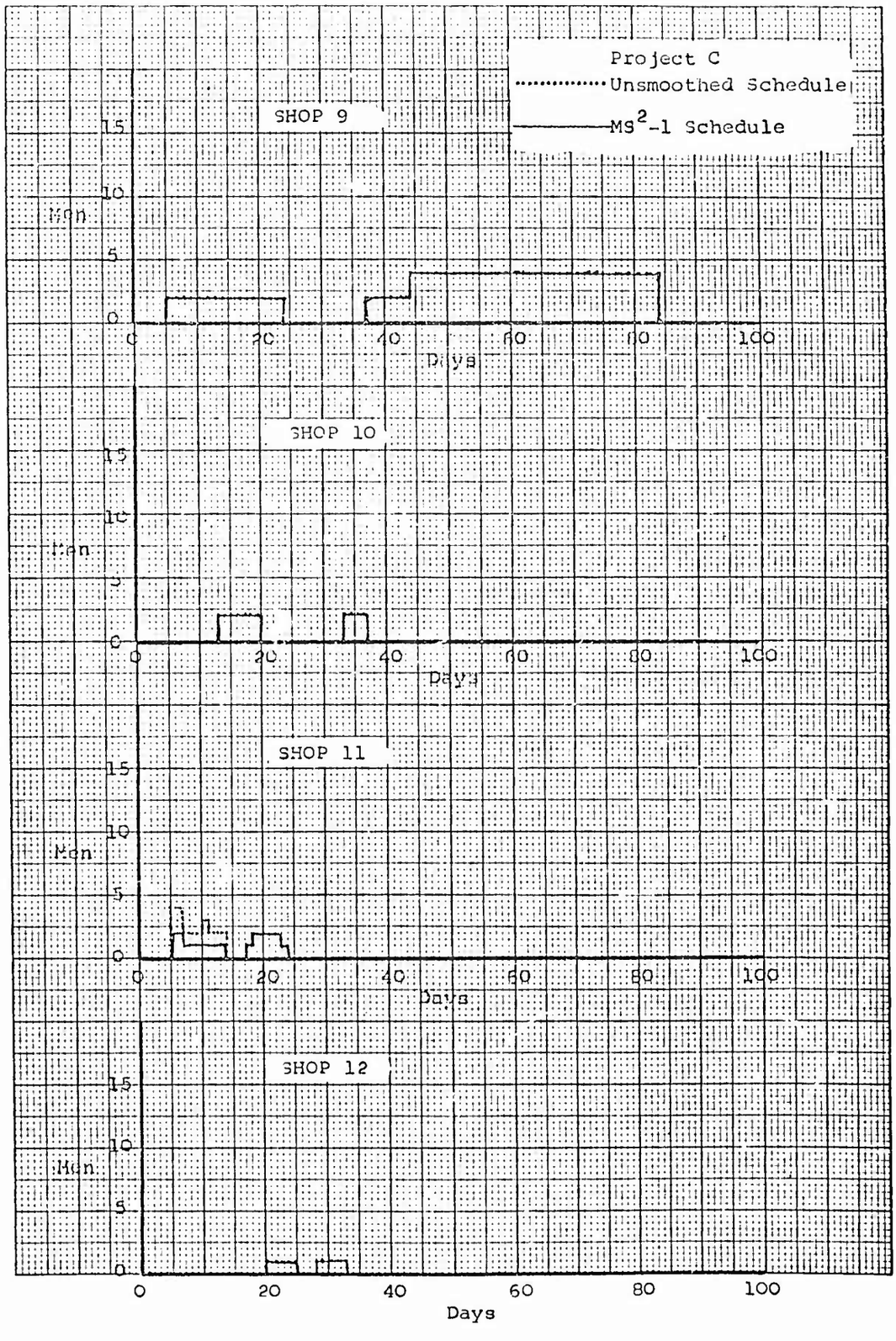


Figure 10 (cont.)



Peak resource requirements were considerably reduced, as is evident in the following summary of data:

	Maximum Resource Requirements											
Shops	1	2	3	4	5	6	7	8	9	10	11	12
All jobs at ES	12	3	20	7	8	12	14	8	4	2	4	1
After smoothing	7	3	13	4	7	7	12	4	4	2	2	1

(Computer running time was 15 minutes.)

After the above results were obtained, the project was scheduled by the SPAR-1 model. Variable crew sizes were allowed when job times could reasonably be expected to be a function of manpower assigned. When determined by technical requirements, job times were held constant (e.g., the job of cement pouring). Search Routine 2 (decreasing shop limits) was used, and initial resource levels were adequate to meet the maximum crew size of any job. Successive iterations of the model produced the following results.

Table 5

SPAR-1 Applied to Project C: 131 Jobs

	Shop Limits												Due	Total[1]
Iteration	1	2	3	4	5	6	7	8	9	10	11	12	Date	Cost
1	12	3	20	7	8	12	14	8	10	2	4	1	72	249,700
2	11	3	19	5	7	11	13	7	9	2	3	1	78	248,800
3	10	3	18	5	7	10	12	6	8	2	2	1	80	234,400
4	9	3	17	4	7	9	12	5	7	2	2	1	79	216,300
5	8	3	16	4	7	8	12	4	6	2	2	1	77	197,900
6	7	3	15	4	7	7	12	4	5	2	2	1	81	197,800
7	7	3	14	4	7	7	12	4	4	2	2	1	100	238,600
8	7	3	14	4	7	7	12	4	5	2	2	1	81	196,200
9	7	3	13	4	7	7	12	4	4	2	2	1	100	236,600
10	7	3	13	4	7	7	12	4	5	2	2	1	81	194,600

1 Total costs are based on overhead charges of $200/day and shop hourly rates as follows (in order, beginning with shop 1): 4.50, 5.00, 2.50, 3.00, 2.25, 4.50, 4.50, 25.00, 4.50, 4.00, 4.00, 4.50.

Note that iteration 9 has shop limits identical to those resulting from the MS^2-1 application, but that the due date is 100 instead of 94. (Repeated iterations of the SPAR-1 program with these shop limits failed to produce a schedule with a due date lower than 100.) Thus the MS^2-1 schedule for this set of shop limits is superior to (i.e., shorter than) the SPAR-1 schedule. However, the SPAR-1 program was able to find several less expensive combinations of shop levels and due dates (based on the given cost parameters), all with earlier due dates than that produced by the MS^2-1 program. For example, iteration 10 uses the same shop levels as the MS^2-1 program, except that the resource level in shop 9 is increased from 4 to 5 men. The resulting due date is 91--compared to the MS^2-1 schedule due date of 94. As a result, the project cost based on the SPAR-1 schedule ($194,600) is considerably lower than that based on the MS^2-1 schedule ($222,200). Manpower loading charts comparing the MS^2-1 schedule with the SPAR-1 schedule (10th iteration) are shown in Figure 11.

Machine time per iteration was about 4 minutes, including schedule print-outs. Computation time was increased, relative to Projects A and B, not only because of the greater number of jobs involved, but also because of the longer schedule length and the presence of multi-skill jobs.

Evaluation of Results: Project C

Unfortunately, it was impossible to compare the schedules produced by our computer models with the actual schedule employed by the contractor for Project C, for two reasons: changes were made in the project after the original plans (which we used)

Figure 11

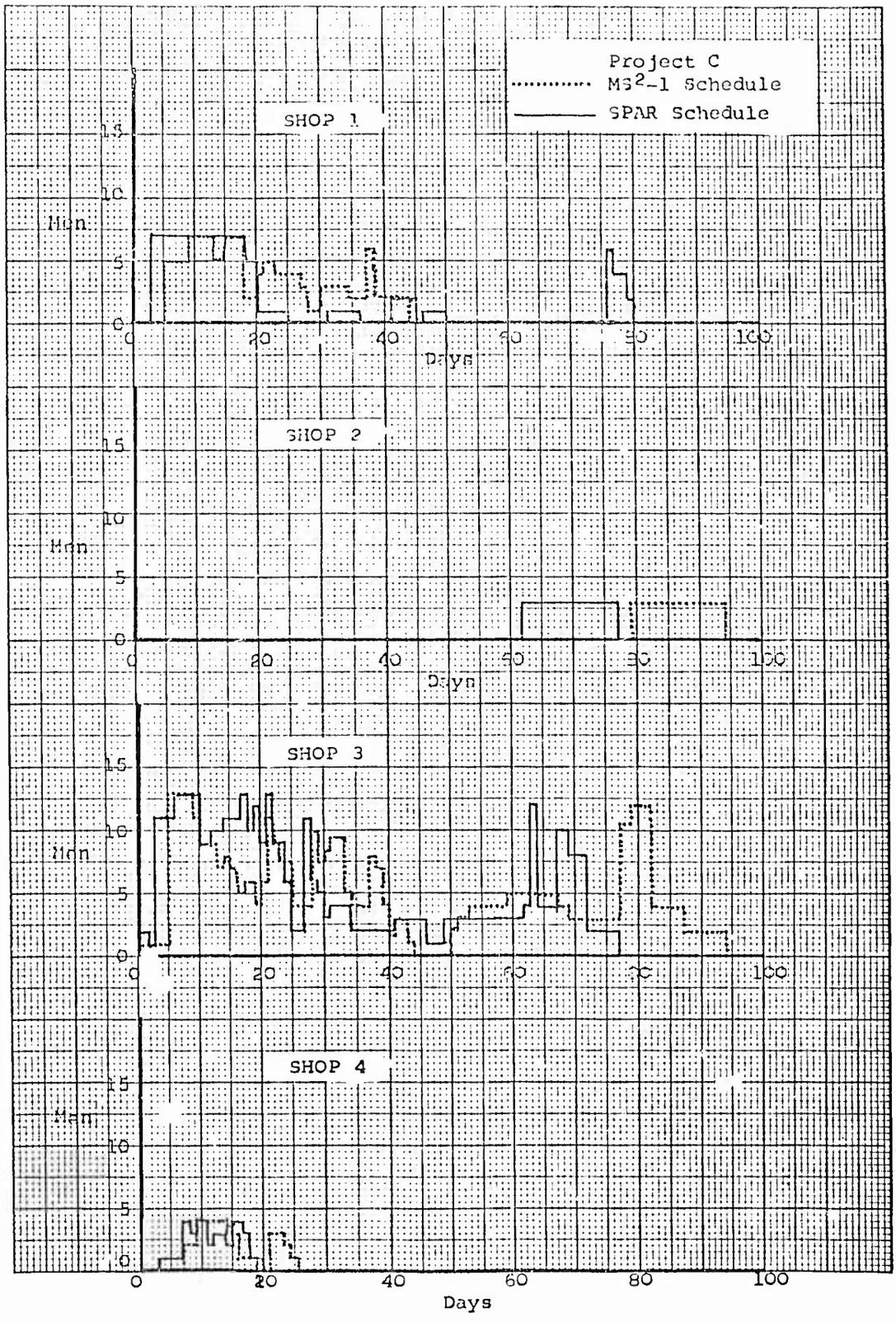

Project C
MS2-1 Schedule
SPAR Schedule
SHOP 1
SHOP 2
SHOP 3
SHOP 4
Men
Days
0
20
40
60
80
100
5
10
15

Figure 11 (cont.)

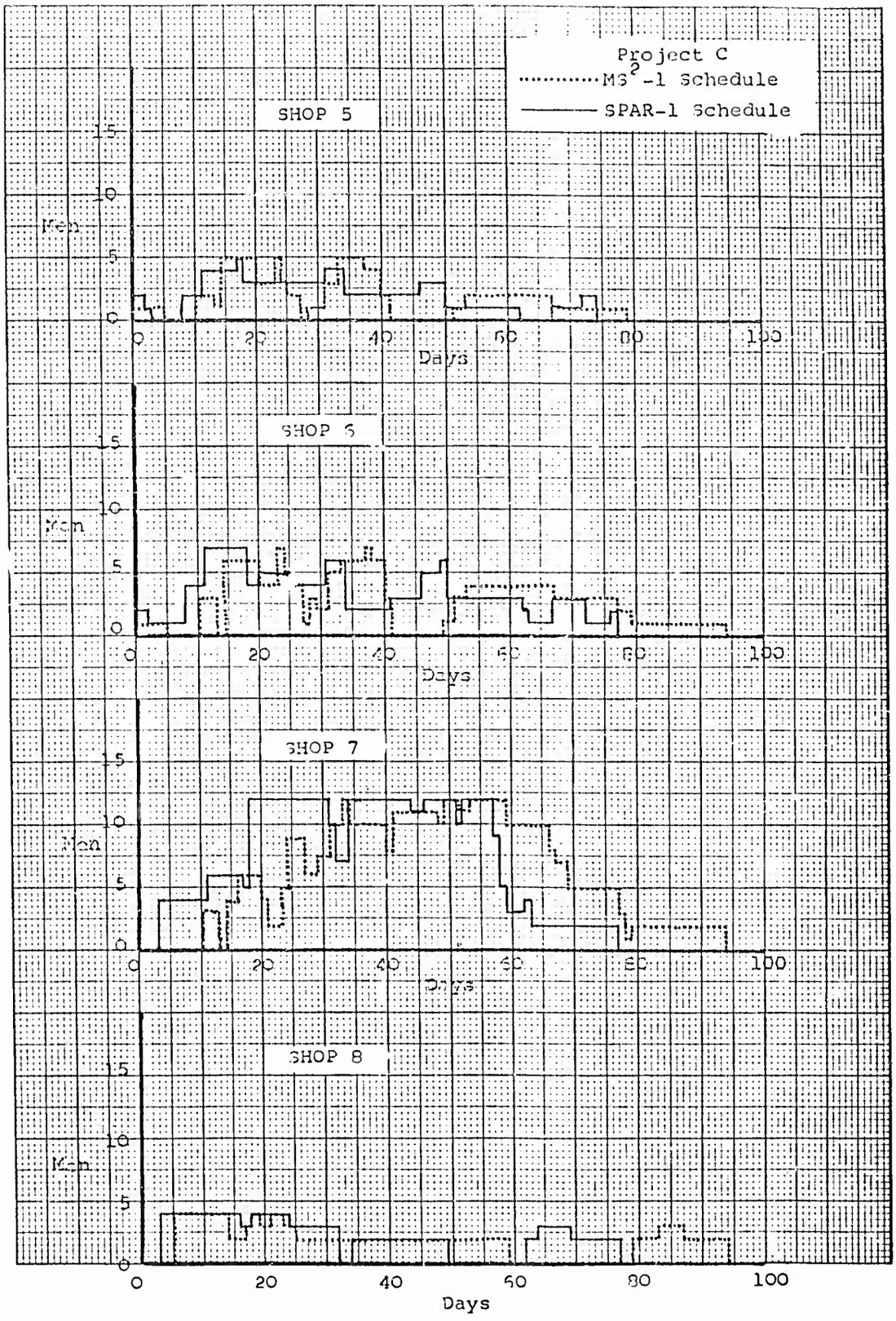
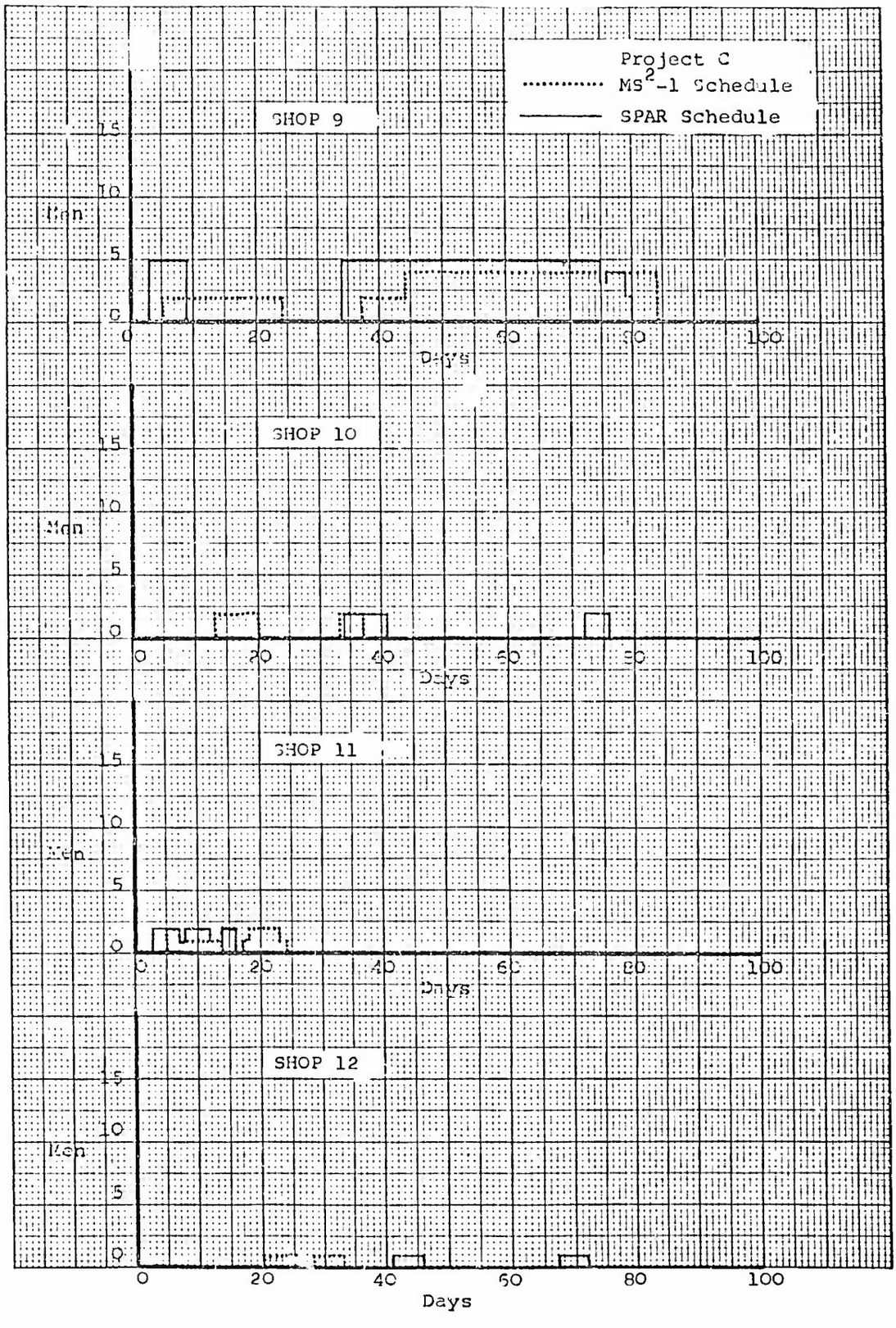


Figure 11 (cont.)



were drawn, and adequate records of actual work performance were not kept. However, some measure of the program's effectiveness was obtained from the contractor's reactions to the schedules they produced. The contractor had used two (and sometimes, three) schedulers for a period of two weeks hand-smoothing jobs in the project. Thus the computer schedules were prepared much more rapidly and at less expense, even when data preparation time necessary for the computer models was considered.[1] The contractor was further impressed with the extent of smoothing which the model achieved---especially on Resource No. 8, whose peak of 8 units unsmoothed was reduced to 4. This resource involved the use of large and expensive machines (e.g., bull-dozers, cranes, tractors, etc.) which had to be rented for the period of use, including short periods of inactivity. Thus if peak requirements for the resource could be reduced, and more intensive (i.e., less interrupted) use made of the machines, considerable savings would result. Not only did the computer models reduce the peak resources required, but (in the case of the SPAR-1 applications) they also indicated how the project length could be appreciably shortened by utilizing variable crew sizes on some jobs---with little or no increase in maximum resource requirements. To the extent that a schedule is shortened with no increase of rescurce levels, resources are used more intensively and idle time is minimized.

The contractor indicated his desire to use one of the computer models for scheduling a much larger, multi-million dollar project. It would be in such an application that the

---

1 See p. 114.

advantages of a computer approach would be most notable. Hand smoothing of a relatively small project--like Project C--is feasible; but the difficulties of manual smoothing increase exponentially with the size of the project and they greatly reduce the potential effectiveness of this approach on large projects. A computer is needed to explore, in a reasonable length of time, the many possible combinations of job start times and crew sizes for such projects.

Project D

By far the largest of the projects scheduled by the computer models was Project D, containing 691 jobs performed in 19 shops. The project consisted of jobs required for month-end closing of the accounting records in a large manufacturing company. Time periods scheduled were one-half hour long, and the schedule horizon was six days, or 288 time periods. The project was complicated by the fact that resource levels varied over the scheduling period; some shops (i.e., manpower skill groups) worked one shift only, while others worked two or three, or irregularly. Shop limits often varied from day to day or from shift to shift (and in some cases during a shift). As we noted in Chapter 3, when resource levels are irregular, the concepts of critical path, critical sequence and job slack lose their normal meaning. The MS^2-1 model smooths resource levels by shifting jobs with positive slack; implicit in the slack calculations is the assumption of constant resource limits over the scheduling period. Consequently MS^2-1 could not be applied to Project D. Although the job-selection subroutine in the SPAR models is based on job-slack calculations (which

in turn are based on technological orderings only), such calculations affect only the priority of jobs to be scheduled; actual job scheduling is done only if resources are available on the day being considered. Thus the SPAR models can readily schedule projects in which resources are not only limited, but variable as well.

The schedule resulting from the application of SPAR-1 to Project D is not reproduced here, because of the size of the project and length of the schedule, but the results may be summarized. The length of the schedule was 250 days and the computation time 51 minutes (including 6 minutes for printing out the schedule). Given the job requirements and resource restraints of the project, the schedule was as short as possible. If unlimited resources were available, the project could be scheduled i 186 time periods. With the limits as given, however, each of two jobs on the critical path was delayed 32 time periods (i.e., two 8-hour shifts) by the total unavailability of resources during those periods.

Because of the rather long computational time and the fact that an optimal schedule for the given resource limits had been found on the first run, only two runs were made. Both resulted in the same due date, although start times of some jobs differed in the two schedules.

Evaluation of Results: Project D

As with Project C, time estimates for jobs in Project D did not accurately reflect subsequent experience, so that comparison of the computer schedule with the actual schedule was of little worth. The significance of the results from applying

SPAR-1 to the project may be measured, rather, in terms of its ability to handle a comparitively large project with complex scheduling constraints (e.g., variable resource limits and variable crew sizes) in a reasonable length of time. The 51 minutes required for a single run on the computer could be reduced significantly by more efficient programming (and possibly by the use of a larger computer), permitting the use of multi-runs and the exercise of the probability features of the model. Even with a single, 51 minute run, however, the model produced a minimum length schedule that would have required many man-weeks of manual effort to reproduce by regular scheduling means. We should note, also, that the company which provided the project had abandoned efforts to use PERT and similar techniques for scheduling the project, as the variable resource limits rendered such methods inapplicable.

Problem: What is a "Good" Schedule?

It should be clear by now that our heuristic approach to project scheduling does not guarantee an optimum solution, nor does it provide a means of determining how near a given schedule is to optimum. Only when projects are sufficiently small or constrained to be solved by exhaustive search techniques is it possible to make a precise statement about the optimality (or near optimality) of a heuristic, non-algorithmic solution. For most projects of practical interest, however, this approach to evaluating a schedule is simply not feasible. Even our simplest test project (Project A) has an enormous number of possible schedules; it would probably take many days to find an optimum by exaustive search. (Only when we severely

constrained the project by holding crew sizes and resource levels constant were we able to discover the optimum schedules by this technique, as we noted above.)

Thus we are left with the less satisfying, but surely not unreasonable, method of evaluating our schedules by comparing them with those resulting from current scheduling practices. By this criterion, we will conclude that a schedule produced by our models is "good" if it is measurably better than a schedule for the same project produced by conventional scheduling methods---i.e., if it requires lower resource limits for the same due date, or if it permits an earlier due date with the same resource limits, or if it results in some lower cost combination of due date and resource limits.

Summary of Results

How well do our models perform in generating "good" schedules? The evidence we reported above is limited but encouraging, and we may summarize as follows. In the case where exhaustive search methods could be applied, schedules produced by our models fared well: of the two computer schedules for which optimal solutions were found "by hand," one was optimal and the other nearly so. Our schedule for Project C, although not directly comparable to the schedule actually used, was sufficiently "good" to interest the contractor in further applications of the model on more important projects. In all of the schedules for projects A, B and C generated by our model, resource limits were significantly reduced below the levels required by schedules of the PERT and CPM variety (all jobs scheduled at early start). While users of such scheduling methods

may reduce resource levels by hand-shifting of slack jobs, this juggling becomes an enormous task with large projects and the extent to which it may be used is limited. We may safely conclude that our models generated schedules significantly better than those that would be produced by PERT, CPM, and similar techniques---methods that are widely used today. Additionally, SPAR-1 was able to find an optimum schedule (in terms of due date) for Project D--a project which could not be scheduled at all by present PERT-type programs, because of the variable resource limits which had to be considered.

Thus we conclude from our experience with the scheduling models that they do produce "good" schedules, but our evidence is limited to a few examples. We would profit from more experience in applying the models to a variety of project types and sizes, and from additional comparisons of the schedules produced with those resulting from current practice.

Chapter 6

CONCLUSION

In this final chapter, we point to some of the economic and practical advantages of our scheduling models, compare the heuristic procedures we used in our programs with those which have been applied to related scheduling problems, and make some suggestions for future work. We close with a summary of what we consider to be the main contributions of our study to the problem of large project scheduling with limited resources.

Economic Feasibility of the Models

Although we have not made a detailed cost-feasibility study, our preliminary calculations indicate that the costs of applying the models (including data preparation costs and computer time) to projects of the size we have considered is economically attractive, as compared to manual methods of scheduling.

For example, the schedule for project C was generated by MS^2-1 in 15 minutes on the G-20. If we wished to obtain the refinements resulting from the SPAR-1 model, we would add to this time an additional 10 minutes for a SPAR-1 schedule (with the search routine abbreviated by starting with the output data of the MS^2-1 schedule), thus totaling 25 minutes or about $85 in machine time. (This could be reduced considerably by more efficient programming of the models.) Keypunching of data required two to three hours, bringing the total cost to something under $100. Converting computer output to job orders and other forms operationally useful would require additional time; but presumably much of this work could be done by the

computer itself, through proper programming of output, at little additional expense. By comparison, the two (and sometimes three) schedulers assigned to hand-smooth jobs in the project spent more than a man-month doing so. Their salary expense along was over $600. Thus the computer scheduling was considerably less expensive, required a small fraction of the time, and resulted in better smoothing (according to the contractor), as compared with the conventional methods.

Computational time appears to increase roughly in proportion to $(n \cdot d)$, where n is the number of jobs and d the number of days in a schedule. If this relationship may be safely extrapolated to larger projects, then the models would still be economically feasible for scheduling such projects, as conventional scheduling expenses would be expected to increase at least by the same proportion. However, the size of projects which can be handled by the models is presently limited by computer memory capacity rather than by costs of computer time.

Although difficult to evaluate in dollar terms, the advantages of a model which produces a schedule in a matter of minutes rather than weeks are of considerable worth to a manager of a large project. At the time a project is being planned (or when a contractor is preparing bids), the model could simulate the effects of various resource limits on project costs and due dates, a feat that would be quite difficult and much more time consuming to do by hand. As a project is started and work proceeds, almost inevitably the actual work performed varies in some degree from the most carefully planned schedule, due to unanticipated delays or inaccurate time estimates. Schedules must be updated, and the computer model would permit this to be

done rapidly---perhaps on a daily or weekly basis. Manual re-scheduling, especially of quite large projects, would likely lag behind work being accomplished; each new schedule would be obsolete at its completion. Computer-generated schedules should thus permit closer coordination and control of project activities.

Heuristics for Scheduling Problems

We earlier explored the relationship (and differences) between three scheduling problems---assembly line balancing, job shop scheduling, and large project scheduling. Since attempts have been made to solve all three problems by heuristic programs, it is reasonable to ask if the programs are comparable--if at some level the heuristics are similar---and if in any way they build on each other and collectively contribute to our ability to deal with these ill-structured problems. We will answer these questions by comparing three examples of heuristic programs: Gere's job shop scheduling model [15], Tonge's line balancing model [45], and our own large project scheduling models.

We would anticipate that specific similarities in the heuristics of the programs would exist to the extent that the problems are themselves similar, or have similar features, since the heuristics involved were developed with a high degree of specificity for the problems at hand.[1] Such proves to be true: the heuristics in the job shop and large project programs have more in common with each other than with the heuristics of the line balanceing program. For example, both of the former programs contain rules for giving precedence to jobs with the least slack. (The slack concept has no meaning in the line balancing

1 Such need not be the case with heuristic methods. See Newell, Shaw and Simon on the General Problem Solver [36].

problem.) The Look Ahead heuristic in the job shop program is designed to avoid future conflicts in the demand for a resource which result from a current decision that otherwise appears optimal. By comparison, the Borrow and Reschedule heuristics of the SPAR program attempt to resolve current conflicts in the demand for resources by changing job assignments previously made, that seemed optimal at the time. The intent and effect of these heuristics in both programs is much the same. The Insert rule of Gere's program and the Add-on rule of our own, while different in their approach, both attempt to achieve more intensive use of resources and to minimize idle time. While the SIO rule[1] of the job shop program has no counterpart in our own, it is logically applicable to the large project problem and could be incorporated easily in either the MS[2] or SPAR program.

It is difficult to find, on the other hand, any such close similarities between the heuristics of the above scheduling programs and those of the line balancing problem. In part this is due to the substantial differences in the problems concerned, and in further part by the distinctively different approaches to problem solving represented in the three programs--especially in the line balancing program as compared to the other two.[2]

---

1 SIO stands for Shortest Imminent Operation; by this rule, the job requiring the shortest processing time for its next operation has highest scheduling priority.

2 In this respect, it is interesting to note that Tonge found it desirable to write his computer program in IPL, a symbol manipulation language (without which, he stated, he "would never have attempted mechanizing a line balancing procedure such as this"). The other two programs were written in algebraic languages.

At a more general level of analysis, however, it is possible to point out some similarities in the three programs. Tonge cites a number of characteristics of existing heuristic procedures, two of which have particular relevance for our purposes. The first characteristic he refers to is the

> Factorization of the problem into a number of smaller problems and subproblems (often through means-end analysis), with a corresponding goal-sub-goal organization of behavior.[1]

For example, in his line balancing program, groups of elements are aggregated into single compound elements, creating simpler line balancing problems which are more easily solved. The program assigns groups of available workmen to elements and takes as subproblems those compound elements which have been assigned more than one man.

In a similar vein, we may regard the major goal of the SPAR program as the completion of the project (the scheduling of all jobs) within the constraints of limited resources. This goal is factored into the smaller problems of scheduling subsets of jobs that become available as the program progresses over a sequence of time periods. If, on a particular day, some job j cannot be scheduled for lack of resources, the program sets up a second-level subgoal of providing the necessary resources, through the Borrow and/or Reschedule routines. If these attempts fail, the (sub)goal of scheduling job j on day d is modified.

An analogous example may be found in Gere's job shop program, in which the goal of minimizing costs of late jobs is factored into the subgoals of utilizing available machine capacity each period, by scheduling individual jobs according to some priority rules.

---

1 See [45, p. 17].

The second characteristic Tonge cites is the

> Use of cues in the enrivonment to determine the particular behavior evoked from a wide set of possible alternatives available to the program, that is, a high degree of interdependence between the specific problem (from a more general class) being considered and the particular problem-solving methods used.[1]

He notes that the methods for selecting elements to shift between groupings in the assembly balancing program depend on the particular characteristics of the groupings. Similarly, in our project scheduling model (SPAR), the behavior evoked in selecting a job to be scheduled depends on the precedence relationship of the job to other jobs in the network and on the current availability of resources. Critical jobs evoke a different heuristic than jobs with slack, for example, and the pattern of jobs scheduled by a given time period determines whether the Add-on, Borrow, and Reschedule routines are called into play at that period. And in Gere's program, the effects of job selection priority rules are modified by a number of "overriding" heuristics which take cues from the immediate environment of the schedule. For example, the Alternate Operation heuristic considers the effect that scheduling one job has on others that compete for the same machine; and the Look Ahead heuristic attempts to anticipate conflicts that will be created in the future by the present scheduling of a given job.

We could cite further examples, but these serve to illustrate the point which we now make. While it is possible to find quite similar heuristics in some programs (especially if the problems dealt with are related), and while the experience with such heuristics may be transferable in some degree fom one problem to another, it is in terms of the general characteristics of

---

1 op. cit.

heuristic programs that one is most apt to find helpful guides to the solving of specific problems by heuristic procedures. We may answer our earlier question by concluding that the three programs examined contribute individually, rather than collectively, to our ability to deal with the problems they were designed to solve; but that, collectively, they illustrate and give us further experience in the use of general heuristic procedures for problem solving.

Future Work

The results of our work, as could be expected, point to additional areas of research that seem promising for the further development of large project scheduling methods. We suggest the following as logical and potentially fruitful extensions of the present study:

1) Completion and Testing of SPAR-2. The theoretical considerations of Chapter 3 open up new avenues of project scheduling to explore. SPAR-2 represents one approach to scheduling that utilizes the notion of "critical sequence;" we are hopeful that it will prove to be a fruitful approach, and intend to continue the development and testing of this model. Many other approaches, of course, are possible.

2) Additional Decision Rules. SPAR-1 employs a single decision rule for selecting jobs to be scheduled each time period (modified by the probability feature). Other rules could easily be substituted (e.g., pick first the job having the most successors using "bottleneck" resources), and their effects studied. Thompson and Fisher [12] report that probabilistic combinations of two decision rules yield the best results in scheduling problems they have studied.

3) Learning Features. The use in the program of several decision rules, various parameters, and probability constants (see Chapter 4) suggests that the program can be written so as to modify itself as a result of its experience in repeated iterations on a project. Thus the program could successively explore the effects of changes in a given decision rule or parameter and alter itself to use the decision rules or parameters that yield the best schedules.[1] Two problems can be anticipated: small projects would have to be employed, so that scheduling time is short enough to base "learning" on more than limited experience. More important, perhaps, is the problem of generalization. Could "learning" experience on one project be extended to a different project? Would the particular decision rules and parameters which proved successful on a given project be equally successful on all projects, or would the learning process have to start anew with each project studied?

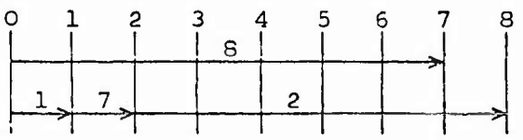
4) Improved Search Routines. The Search routines for SPAR-1 described in Chapter 4 worked reasonably well on the projects tested, but we can think of improvements that would be desirable to work out. Essentially the routines operate by taking trial-and-error samples from the space of all schedules the model is capable of generating. At present, a search procedure may be halted prematurely when it reaches a local optimum; methods of reaching beyond these points to more favorable schedules would add to the effectiveness of the search routines. Secondly, a means of moving more rapidly towards optimum points (and then

---

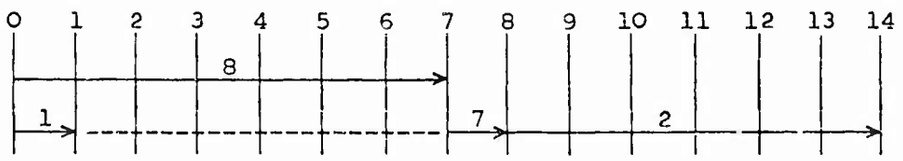
1 Of particular interest is the work of Fischer and Thompson, who explored the subject of "learning" in the job-shop scheduling problem [12].

searching more methodically in their immediate vicinity) would reduce the number of iterations needed in the search procedure and improve its efficiency--extending the potential use of the tool to larger projects.

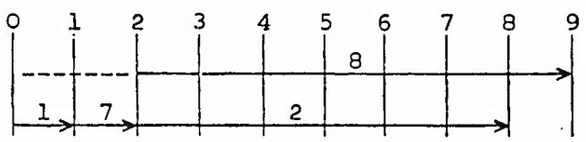
5) Improved Reschedule Routine. It is possible to think of projects in which the Reschedule routine in SPAR-1 fails to postpone a job that, if rescheduled, would lead to an earlier due-date. For example:



With a resource limit of 10, SPAR-1 would generate the following schedule:



The due date of 14 compares unfavorable with that of the optimum solution



which has a due date of 9. A slightly more sophisticated search routine would discover such situations and select the best order for jobs to be scheduled.

6) Additional Emperical Studies. We have already mentioned the need to test further the models and study results from their applications to a variety of actual projects. The above suggested expansions of scheduling heuristics in the models would also add to the need for additional empirical studies.

Summary

Our study was prompted initially by the need we saw for more effective methods of large project scheduling. Because most recent advances in this field have paid little explicit attention to the problem of limited resources, we devoted much of our effort to developing a theoretical system for dealing with project scheduling when resources are scarce. We classified different types of schedules according to their properties, defined operations useful for generating and altering schedules, extended the concepts of critical path analysis to the limited resource case--including the useful notion of slack, developed the concept of a critical sequence, and proved several theorems regarding its existence in certain types of schedules. In so doing we related our work with that of Giffler and Thompson for the job-shop problem, and showed how the latter could be considered a special case of the large project problem.

Additionally, we developed some scheduling models for large projects using a heuristic approach and taking advantage of the computational powers of a large computer. The models employ several of the concepts mentioned above and were designed to handle many features of scheduling problems not previously considered (or incorporated all in one model)--e.g., variable crew sizes, limited (and in some cases, variable) resources, multi-resource jobs, costs of regular and idle time, overtime, and due date penalties, the possibility of trade-offs between resource limits and due date, multiple projects, and so forth. Although applications of the models have been somewhat limited to date, the results from these applications have been gratifying. In trial runs on several projects, the models generated

schedules notably better than those that would be generated by PERT and related programs--methods widely employed in the United States today and currently required by the Government for use on many large defense contracts. We are confident that our computer models have measurably advanced the techniques of large project scheduling.

While there are many improvements that can be made in the programming and logic of the models (and we have made some specific suggestions in this regard), they are presently feasible--economically and operationally--for use on projects of medium-large size (about 1,000 jobs). Our experience has demonstrated to our satisfaction that a heuristic approach to project scheduling, aided by the computational power of a digital computer, can contribute in an important way to solving some of the complex scheduling problems of large project management.

A Final Comment

Although our attention has been focused mainly on industrial problems, it would be appropriate as we conclude our study to emphasize that the concepts we have developed and the models tested are not limited to an industrial setting. The allocation of limited resources is a ubiquitous economic problem, and one can find examples of large project scheduling problems in many guises. We may note one as an example. Economic planning for the development of a country's resources and productive capacity is essentially a problem of allocating limited resources over a period of time. For many underdeveloped countries, the building of an industrial society may be viewed as a project consisting of a partially ordered set of "jobs:" steel mills come before

appliance factories, school buildings before super highways, tractors before TV sets, and so forth. The concepts of Chapter 3 would enable the economic planner to analyze the precedence network of such "jobs" in the context of limited resources, and the models of Chapter 4, drawing on these concepts, would assist him in finding feasible allocations of these resources over a planning horizon. While his problems are much more complicated than those of an industrial project planner and his "project" less clearly delineated, the tools and techniques discussed here should still be useful to him--and to others concerned with the planning of large projects where available resources are limited.

GLOSSARY OF SYMBOLS

A	Resource availability matrix; dimension m x z
a_{sd}	A component of A; the resources available in shop s on day d
A_s	Vector of resource availability, by days, in shop s; corresponds to row s in A
a_d	A component of A_s
AS_j	Assigned start of job j
AF_j	Assigned finish of job j
b_s	A parameter in the SPAR model
C	Set of jobs concurrent to a local set
c	An element of C
CS	Set of jobs comprising a critical sequence
cs	Crew size (i.e., resources required for a job)
cs_m	Minimum crew size
cs_M	Maximum crew size
cs_n	Normal crew size
d	Day
ES_j	Early start of job j
EF_j	Early finish of job j
F	Project finish date
G	Set of jobs constraining a local set
g	An element of G
i	Iteration variable
J_l	Set of left-justified schedules for project X
J_r	Set of right-justified schedules for project X
j	Job
K	Overhead expenses and due date penalty, per day (SPAR model)

L	Local set of jobs
l	An element of L
LS_j	Late start of job j
LF_j	Late finish of job j
m	Number of shops or resource groups required by jobs in X
n	Number of jobs in X
P_j	Set of jobs which are immediate predecessors of j
Q	Resource requirements vector; dimension m x z
q_{sd}	A component of Q; the resources required in shop s, day d
Q_s	Vector of resource requirements, by days, in shop s; corresponds to row s in Q
q_d	A component of Q_s
R_s	Set of jobs comprising a resource sequence in shop s
r_s	Remaining resource in shop s on day d (SPAR model)
S	Project start time (usually S = 0)
S_j	Set of jobs which are immediate successors of j
s	Shop (resource group)
T	Set of jobs comprising a technological sequence
t	Time required to complete a job
v	Overtime premium factor (SPAR model)
w_s	Average wage rate in shop s
X	A set of jobs comprising a project
x	A schedule which is a member of J_l
y	A schedule which is a member of J_r
Z	A span of time
z	The number of days in a Schedule Chart (the length of the schedule)
<<	Is an immediate predecessor of

BIBLIOGRAPHY

1. Alford, L. P., and John R. Bangs, (Editors), Production Handbook, The Ronald Press Company, New York, 1951.

2. Alpert, L., and D. S. Orkand, "A Time Resource Trade-Off Model for Aiding Management Decisions," Technical Paper No. 12, Operations Research Inc., Silver Spring, Maryland (1962).

3. Avots, Ivars, "The Management Side of PERT," California Management Review, Vol. 4, No. 2 (Winter 1962).

4. Bowman, E. H., "Assembly-Line Balancing by Linear Programming," Operations Research, Vol. 8, No. 3 (May - June, 1960).

5. Bowman, E. H., "The Schedule-Sequencing Problem," Operations Research, Vol. 7, No. 5 (September-October 1959).

6. Charnes, A., and W. W. Cooper, "A Network Interpretation and a Directed Subdual Algorithm for Critical Path Scheduling," Journal of Industrial Engineering, (July-August 1962).

7. Clark, C. E., "The Optimum Allocation of Resources Among Activities of a Network," Journal of Industrial Engineering, (January-February 1961).

8. Clarke, Roderick W., "Management Systems for the Economic Accomplishment of System Development Projects," Graduate School of Business, Stanford University, Stanford, California (July, 1962).

9. Clarkson, Geoffrey P. E., Portfolio Selection: A Simulation of Trust Investment, Prentice Hall, Englewood Cliffs, N. J., 1960.

10. Clarkson, G. P., and A. H. Meltzer, "Portfolio Selection: A Heuristic Approach," Journal of Finance (December, 1960).

11. Davis, Fischer and Marvin, "PERTCO I (PERT Plus Cost) A Report of Progress, Capability and Conclusions to Date," Douglas Aircraft Company, Santa Monica, California, (October, 1961).

12. Fischer, Henry, and Gerald L. Thompson, "Probabilistic Learning Combinations of Local Job Shop Scheduling Rules," O.N.R. Research Memo No. 80, Graduate School of Industrial Administration, Carnegie Institute of Technology, (February 1961).

13. Ford, L. R., and D. R. Fulkerson, "A Simple Algorithm for Finding Maximal Network Flows and an Application to the Hitchcock Problems," Canadian Journal of Mathematics, 9, (1957).

14. Fulkerson, D. R., "A Network Flow Computation for Project Cost Curves," Management Science, Vol.7, No. 2, (January 1961).

15. Gere, William S. Jr., "A Heuristic Approach to Job Shop Scheduling," Doctoral Thesis, Carnegie Institute of Technology (1962).

16. Giffler, B. and G. L. Thompson, "Algorithms for Solving Production Scheduling Problems," Operations Research, Vol. 8, No. 4, (July-August, 1960).

17. Gomory, Ralph E., "Outline of an Algorithm for Integer Solutions to Linear Programs," Bulletin of the American Mathematical Society, 64, (September, 1958).

18. Grossman, H., "The Development of SCANS - A Network System for Management Control," Systems Development Corporation, Santa Monica, California (February 1961).

19. Jackson, J. R., "A Computing Procedure for a Line Balancing Problem," Management Science, Vol.2, No. 3 (April, 1956).

20. Karg, Robert, and G. L. Thompson, "A Heuristic Program for the Traveling Salesman Problem," Graduate School of Industrial Administration, Carnegie Institute of Technology, (September 1962).

21. Kelley, J. E., Jr., "Critical-Path Planning and Scheduling: An Introduction," Mauchly Associates, Inc., Ambler, Pennsylvania (1959).

22. Kelley, James E., Jr., "Critical-Path Planning and Scheduling: Mathematical Basis," Operations Research, Vol. 9, No. 3 (May-June 1961).

23. Kelley, J. E., Jr., and M. R. Walker, "Critical-Path Planning and Scheduling," Proceedings of the Eastern Joint Computer Conference, Boston, December 1-3, 1959.

24. Koepke, Charles A., Plant Production Control, John Wiley & Sons, Inc., New York, 1961.

25. Kuehn, A. A., and M. Hamburger, "A Heuristic Program for Locating Warehouses," Graduate School of Industrial Administration, Carnegie Institute of Technology, (September 1962).

26. Levy, F. K., G. L. Thompson, and J. D. Wiest, "An Introduction to the Critical Path Method," Factory Scheduling, (edited by J. F. Muth and G. L. Thompson), Prentice Hall, Englewood Cliffs, N. J. (forthcoming).

27. Levy, F. K., G. L. Thompson, and J. D. Wiest, "Mathematical Basis of the Critical Path Method," Factory Scheduling, (edited by J. F. Muth and G. L. Thompson), Prentice Hall, Englewood Cliffs, N. J. (forthcoming).

28. Levy, F. K., G. L. Thompson, and J. D. Wiest, "Multi-Ship, Multi-Shop Workload - Smoothing Programs," Naval Research Logistics Quarterly (March 1962).

29. Levy, F. K., and J. D. Wiest, "A Simulation Approach to Determining the Stochastic Characteristics of a Project," Graduate School of Industrial Administration, Carnegie Institute of Technology (1963).

30. Malcolm, D. G., J. H. Roseboom, C. E. Clark, and W. Fazar, "Application of a Technique for Research and Development Program Evaluation," Operations Research, Vol. 7, 1959.

31. Manne, Alan S., "On the Job Shop Scheduling Problem," Operations Research, Vol. 8, No. 2 (March-April 1960).

32. McGee, A. A., and M. D. Markarian, "Optimum Allocation of Research/Engineering Manpower Within a Multi-Project Organization Structure," IBM File Number 61-907-171, International Business Machines Corporation, Federal Systems Division, Owego, New York (1961).

33. Miller, Robert W., "How to Plan and Control With PERT," Harvard Business Review, Vol. 40, No. 2 (March-April 1962).

34. Moore, Franklin G., Production Control, McGraw-Hill Book Company, Inc., 1951; Revised Edition 1959.

35. Muth, John F., "The Effect of Uncertainty in Job Time on Optimal Schedules," C. N. R. Research Memorandum No. 88, Graduate School of Industrial Administration, Carnegie Instituteof Technology (January 1962).

36. Newell, A., J. C. Shaw, and H. A. Simon, "Report on a General Problem Solving Program," Proceedings of the International Conference on Information Processing, UNESCO, Paris (1959).

37. Newell, A., and H. A. Simon, "The Logic Theory Machine," Transactions on Information Theory, Vol. IT-2, No. 3 IRE (September 1956).

38. Newell, Allen, and Herbert A. Simon, "Heuristic Programs and Algorithms," C.I.P. Working Paper No. 39, Graduate School of Industrial Administration, Carnegie Institute of Technology (May 1962).

39. Reitman, Walter R., "Heuristic Programs, Computer Simulation and Higher Mental Processes," Behavioral Science (October 1959).

40. Sauer, Ray N., "Least Cost Estimating and Scheduling (LESS) - Scheduling Portion," International Business Machines Corporation, Houston, Texas (February 1961).

41. Simon, Herbert A., and Allen Newell, "Heuristic Problem Solving: The Next Advance in Operations Research," Operations Research, Vol. 6, No. 1 (January-February, 1958).

42. Simon, H. A., The New Science of Management Decision, Harper & Bros., New York, 1960.

43. Snyder, Charles J., Jr., "PECOS - Project Evaluation and Cost Optimization System," International Business Machines Corporation, Akron, Ohio (August 1962).

44. Tonge, Fred, "The Use of Heuristic Programming in Management Science," Management Science, Vol. 7, No. 1 (October 1960).

45. Tonge, Fred M., A Heuristic Program for Assembly Line Balancing, Prentice Hall, Inc. Englewood Cliffs, N. J., 1961.

46. Voris, L. P., Production Control, Text and Cases, Richard D. Irwin, Inc., Homewood, Illinois (1961).

47. Wagner, H. M., "An Integer Linear Programming Model for Machine Shop Scheduling," Naval Research Logistics Quarterly, Vol. 6, (1959).

48. Anonymous, "Integrated Management Planning and Control Technique (IMPACT)" Lockheed Report 14693, Lockheed Aircraft Corporation, Burbank, California (September 1960).

49. Anonymous, "Nortronics Approach to PERTCO," Nortronics Division of Northrop Corporation, Hawthorne, California (May 1961).

50. Anonymous, "PERT: An Integrated Management Information Control System," Operations Research, Inc., Silver Spring, Maryland (September 1961).

51. Anonymous, "PERT/COST Systems Design, DOD and NASA Guide, Office of the Secretary of Defense, National Aeronautics and Space Administration (June 1962).

52. Anonymous, RAMPS Training Text, C·E·I·R Inc., Arlington, Va. (1962).

53. Anonymous, RAMPS Users Guide, C·E·I·R Inc., Arlington, Va. (1963).

54. Anonymous, USAF PERT COST System Description Manual, AFSC PERT Control Board, Hq. AFSC (SCCS), Washington, D.C. (March, 1963)